AD-A268 375

MENTATION PAGE

OMB No. 0704-0188

FINAL/01 SEP 89 TO 31 OCT 92

DESIGN & DIAGNOSIS PROBLEM SOLVING WITH MULTIFUNCTIONAL TECHNICAL KNOWLEDGE BASES (U)

6. AUTHOR(S)

Professor B. Chandrasekaran

6919/00/DARPA
F49620-89-C-0110

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Ohio State University
Dept of Computer & Information Science
1960 Kenny Road
Columbus OH 43210

8. PERFORMING ORGANIZATION REPORT NUMBER

AFOSR-TR- 93 0575

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFOSR/NM
110 DUNCAN AVE, SUTE B115
BOLLING AFB DC 20332-0001

DTIC
ELECTE
AUG 16 1993
S A D

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

F49620-89-C-0110

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

UL

13. ABSTRACT (Maximum 200 words)

The research goal for the project reported here has been to develop enabling technology for building large knowledge bases designed to support automated and semi-automated problem solving for scientific and engineering domains. The work on analyzing problem-solving tasks, much of it preceding the project reported here, has led to the view that there are distinct, analyzable "Generic Tasks" (GT's) that occur naturally in problem-solving activity. The "Functional Representation Language" (FR), was originally introduced by Sembugamoorthy and Chandrasekaran (1986) as a device representation from which diagnosis-specific knowledge could be derived by a process of knowledge compilation. It was subsequently applied to support other tasks besides diagnosis (most notably forms of qualitative simulation), and used to represent a wide variety of device types.

93 8 13 063

93-18809

14. SUBJECT TERMS

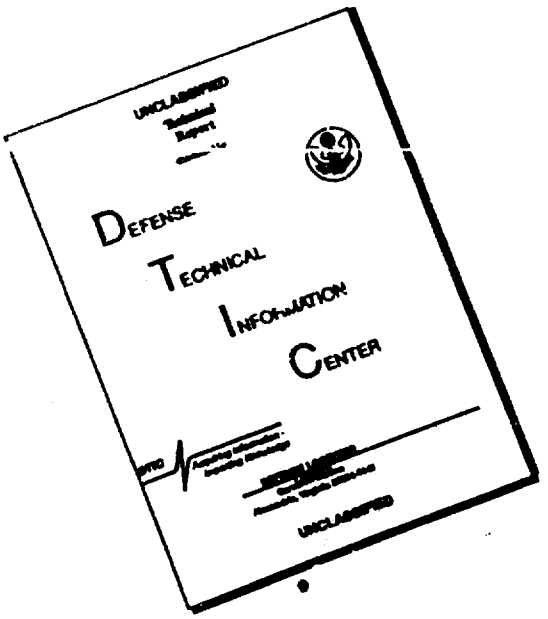
16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR(SAME AS REPORT)

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1993	Final (Oct. 1989 - Dec. 1992)

4. TITLE AND SUBTITLE

Design and Diagnosis Problem Solving with Multifunctional Technical Knowledge Bases

5. FUNDING NUMBERS

DARPA Order No. 6919

AFOSR Contract No. F49620-89-C-0110

6. AUTHOR(S)

B. Chandrasekaran and John Josephson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The Ohio State University Research Foundation
1960 Kenny Rd.
Columbus, Ohio 43210

8. PERFORMING ORGANIZATION REPORT NUMBER

RF 767812/722778

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Office of Scientific Research
Building 410
Bolling Air Force Base, D.C. 20332-6448

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

F49620-89-C-0110

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

14. SUBJECT TERMS

15. NUMBER OF PAGES
224

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

THE OHIO STATE UNIVERSITY

Design and Diagnosis Problem Solving with Multifunctional Technical Knowledge Bases

B. Chandrasekaran and John Josephson
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science

Department of Defense
Advanced Research Projects Agency

DTIC QUALITY INSPECTED 3

DARPA Order No. 6919
Monitored by AFOSR under Contract No. F49620-89-C-0110
Final Report
For the Period: October 1989 - December 1992
RF Project No. 767812/722778

June 1993

Table of Contents

Report Overview

The research goal for the project reported here has been to develop enabling technology for building large knowledge bases designed to support automated and semi-automated problem solving for scientific and engineering domains.

This report has the following sections:

- Task Structure Analysis
- Representing and Reasoning about Devices and Causal Processes
- Visual Reasoning
- Reconfigurable Devices and Other Applications
- References
- Papers and Publications Supported by this Project
- Appendices: Included Papers

Task Structure Analysis

Our work on analyzing problem-solving tasks, much of it preceding the project reported here, has led to the view that there are distinct, analyzable "Generic Tasks" (GT's) that occur naturally in problem-solving activity, and that largely determine:

- the forms of knowledge that will be most useful to store in a knowledge base,
- the problem-solving strategies that will be effective for using that knowledge to solve problems, and
- the knowledge organizations that will make knowledge retrieval efficient.

Some other determiners of forms and organizations of knowledge, and of effective reasoning strategies, are: the nature of the available computational resources for problem solving (parallel or serial, size of short- and long-term memory, processing speed, etc.); the forms in which knowledge is most readily available; and the forms of knowledge that most readily support human interaction, including explanation of reasoning processes and justification of conclusions.

We have distinguished and studied seven "elementary" GTs, with their associated knowledge forms and reasoning strategies:

- **Hierarchical Classification-** Classify an object, event, or situation with respect to a taxonomic hierarchy. A useful control strategy is "establish-refine," i.e., top-down, prune-or-pursue navigation of the hierarchy of classification categories, where each category is associated with knowledge to support Hypothesis Matching.

- **Hypothesis Matching-** Recognize whether a concept applies to a given situation, producing a confidence symbol. Knowledge is usefully organized hierarchically, along lines of abstraction of features. That is, a "feature" may itself be recognized based upon a further set of features. Control may be organized for either hypothesis-driven or event-driven invocation. This generic task has also been called "structured matching," "routine recognition," and "concept matching for relevance."

- **Routine Design-** Construct a design or plan according to specifications, where a family of design plans is known in advance, but how plans and sub-plans fit together must be decided at run time. Useful control strategy: plan selection and refinement. Useful forms of knowledge: designs, design plans (plans for constructing designs), plan steps, subplans, constraints to check, and backtracking plans. [3,4]

- **Knowledge-Directed Data Retrieval-** Retrieve data "intelligently" using indirect inference if necessary (either for automated problem solving or direct human use). Data-related concepts are active entities that reside in networks of semantic relations. At a basic level the control mechanism is indirect inference (attached inference procedures) whereby the active concept that catches a question (because it specializes in a concept that occurs in the question) knows what to do to infer an answer. Specialized forms of attached inferenc[illegible]ake advantage of the semantic relations between the concepts. A part[illegible]arly important use of knowledge-directed data retrieval is for "data-abstraction," the transformation of raw data to forms more useful for problem solving, e.g., inferring normality/abnormality and trends. [33]

- **Abductive Hypothesis Assembly**- Find the best explanation for some body of data (constructing a composite hypothesis if necessary) and estimate confidence. Useful forms of knowledge include: knowledge for finding potential explainers (forms include: cues, hierarchical classification knowledge, causal knowledge), hypothesis matching knowledge for evaluation and screening of potential explainers, knowledge for determining explanatory coverage (e.g. causal knowledge), and interaction knowledge for hypothesis fragments (e.g., incompatibility). Useful control strategy: specialized means-ends with the goals of explaining as much as possible, maintaining consistency, and maximizing confidence. [27, 41, 25, 26, 18]

- **Prediction by State Abstraction**- Predict the effects on a system, given changes to lower-level subsystems. Control usefully proceeds bottom-up through the hierarchy of concepts associated with subsystems.

- **Goal Abandonment**- Decide which goal to abandon when it is judged that not all active goals can be achieved. Useful forms of knowledge: precompiled comparative priorities, goal-subgoal hierarchy. Useful control strategy: from the points in the goal-subgoal hierarchy representing the active goals in conflict, inherit priority knowledge from active supergoals.

Elementary GTs form building blocks for more complex reasoning tasks. A major goal of this project was to analyze the task structures of **design**, **diagnosis**, and **redesign** (finding inadequacies in designs, explaining their emergence, and proposing fixes) [6, 7, 11]. These were selected for special attention because of the theoretical challenges they present, and because systems for assisting with these tasks have an enormous potential for useful applications. Analyzing the structure of problem-solving tasks involves determining appropriate methods for accomplishing them under various conditions, analyzing the subtasks spawned by the methods, and determining the characteristics of the various methods, including their knowledge requirements.

Analyzing design, redesign, and diagnosis problem-solving tasks has led us to a more general view of task decomposition and the generation of subtasks [9]. In particular we have found it to be very useful to distinguish method selection as an important distinctive activity which arises after a problem solving task or subtask has been generated. At that time the relevant available information can be used in selecting the most promising method to be used in carrying out the task. The method selection may

itself spawn additional needs for knowledge and problem solutions. So the recursive decompositic of an overall problem into task and subtask is not always fixed, but is a product of the interaction of how the problem instance, the knowledge available to the system when the task arises, and perhaps some history of efforts at solving the problem.

A *task structure analysis* is a functional decomposition of an information processing task. Such an analysis identifies domain-specific and domain independent aspects of a task and methods for its accomplishment. The task can be feasibly accomplished if a decomposition can be found such that each subtask is feasible, and can be combined with the other subtasks by a feasible method. A task can have one or more methods associated with accomplishing it. Each method is specified in terms of how it uses knowledge and inference to achieve its goals, and in terms of what subtasks (subgoals) it sets up and requires to be achieved before it can succeed. Alternative methods for accomplishing a task may make use of a common subtask. This kind of decomposition can be done recursively until methods which achieve subtasks, but which do not set up additional subtasks of their own, are reached.

We investigated how elementary generic tasks can be composed and integrated, trying to combine the advantages of task-specific architectures (computational efficiency, modularity, knowledge-acquisition, knowledge-base debugging, knowledge-maintenance) with the flexibility and robustness inherent in generalized goal-seeking control of problem-solving. Various approaches to integrating basic generic reasoning tasks have been explored at LAIR in recent years. A discussion of the issues involved is given in B. Chandrasekaran, "Design problem solving: A task analysis" which is included as an Appendix with this report. Additional discussion is available in [11, 19, 22, 23, 24, 40, 42].

Task Analysis of Design

The most common top-level family of methods for design can be characterized as **Propose-Critique-Modify (PCM)** methods. These methods have the subtasks of *proposal* of partial or complete design solutions, *verification* of proposed solutions, *critiquing* the proposals by identifying causes of verification failure, if any, and *modification* of proposals to better satisfy design goals. These subtasks can be combined in fairly complex ways, but the following is one straightforward way in which a PCM method may organize and combine the subtasks.

- Step 1. Given design goal, propose solution. If no proposal, exit with failure.
- Step 2. Verify proposal. If verified, exit with success.
- Step 3. If unsuccessful, critique proposal to identify sources of failure. If no useful criticism available, exit with failure.
- Step 4. Modify proposal and return to 2.

While all PCM methods will need to have some way to achieve the iteration in Step 4 above, there can be numerous variants on the way the methods in this class work. For example, a solution may be proposed only for a part of the design problem, a part deemed to be crucial. This solution may then be critiqued and modified. This partial solution may generate additional constraints, leading to further design commitments. Thus subtasks can be scheduled in a fairly complex way, with subgoals from different methods alternating.

In "Design problem solving: A task analysis," which is included in the Appendix to this report, a detailed discussion is provided of the methods that are available for each of the major subtasks of design problem solving, along with an analysis of the knowledge requirements for each of the methods and control strategies. The This paper also discusses the implications of this analysis for the construction of design knowledge bases.

Task Analysis of Abduction

The concept of abduction has been used to frame the problem of diagnosis, scientific theory formation, natural language understanding, and is a more general framework than classification for describing visual recognition. **Abduction** or **inference to the best explanation** is distinctive kind of inference that goes from data describing something to an explanatory hypothesis that best explains or accounts for the data. Thus abduction is a kind of theory-forming or interpretive inference. Abductive inferences have this form, approximately:

D is a collection of data (facts, observations, givens).
H (hypothesis) explains D (would if true, explain D).
No other hypothesis is able to explain D as well as H does.

Therefore, H is probably true.

The judgment of likelihood in the conclusion depends on:

- How decisively H surpasses the alternatives.
- How good H is by itself, independently of considering the alternatives .
- How much confidence there is that all plausible explanations have been considered (how thorough was the search for alternative explanations).

We can analyze the information-processing task of abduction to have two main subtasks:

- subtask 1) *generating elementary hypotheses,*
- subtask 2) *synthesizing composite explanations.*

The subtask, *generating elementary hypotheses,* has two sub-subtasks:

- sub-subtask 1.1) *evoking*
- sub-subtask 1.2) *instantiating elementary hypotheses*

The sub-subtask 1.2., *instantiate elementary hypothesis,* has two sub-sub-subtasks:

- sub-sub-subtask 1.2.1) *scoring the elementary hypotheses*
- sub-sub-subtask 1.2.2) *determining explanatory coverage.*

This decomposition is very general. A typical abductive conclusion is a composite, and somehow, explicitly or implicitly, there must be some method of choosing which elementary hypothesis to consider, some way of making them specific to the case, and some way of accepting and combining.

A rich and comprehensive book on abduction that describes research progress made at LAIR over several years has been completed, and will soon be published: Abductive Inference: Computation, Philosophy, Technology, Ed. J. and S. Josephson, Cambridge U. Pr. (forthcoming).

Task-Structure Analysis and the Construction of Technical Knowledge Bases

In "Task-Structure Analysis for Knowledge Modeling," which is included in the Appendix, we discuss in detail how the task structure provides a road map for knowledge modeling and hence for knowledge representation. The generic methods in the task structure require specific types of

knowledge, that can be supported by task- and method-specific high-level languages.

Prediction

We have devoted much effort to identifying knowledge useful for prediction, because prediction is one of the most pervasive subtasks generated by processes of design checking and diagnostic hypothesis evaluation. We have been concerned with common-sense simulation of the world, and its integration with more technical and mathematical simulation methods, as would be found in an engineer's problem solving. A treatment of some of the foundational issues is given in: B. Chandrasekaran, "QP is More Than SPQR and Dynamical Systems Theory: Response to Sacks and Doyle," which appeared in Computational Intelligence, and is included in the Appendix with this report.

Representing and Reasoning About Devices and Causal Processes

The "**Functional Representation Language**" (**FR**), was originally introduced by Sembugamoorthy and Chandrasekaran (1986) as a device representation from which diagnosis-specific knowledge could be derived by a process of knowledge compilation. It was subsequently applied to support other tasks besides diagnosis (most notably forms of qualitative simulation), and used to represent a wide variety of device types.

Humans understand how the functions of a device result from causal processes in which the components and subsystems of the device play various roles. We have sought to imitate human abilities to reason causally by pursuing "machine understanding of how things work." Making distinctions between device structures, functions, and the causal processes that support functions, is at the heart of the FR, consequently we have used FR as a representational point of departure for these investigations. The main goals have been to devise structures for knowledge organization supporting automatic and manual navigation of multiple levels of functional and causal detail, for a wide range of types of devices and causal processes; and to determine the types of knowledge that are needed to support the problem-solving abilities that normally, in humans, result from understanding how a device works.

FR represents devices (abstract or physical), components, functions, and causal processes - and coherently ties these representations together by connecting device functionality to the causal processes by which functions are achieved. Since its introduction by Sembugamoorthy and Chandrasekaran, FR has been developed and applied by Chandrasekaran, Sticklen, Keuneke, Josephson, Goel, Allemang, Weintraub, Punch, DeJongh, Darden, Moberg, Iwasaki, and others. A very diverse set of devices has been represented in FR, including: electro-mechanical devices [43] [45], biological processes [15] [50] [40], non-AI computer programs [2], knowledge-based systems [51], manufacturing fabrication processes [48], landscape-level ecological systems [49], logistic plans [12], and parts of chemical-processing plants [30] [17]. Inference procedures have been developed to use these representations for case-based design and redesign [17], automatic compilation of diagnostic knowledge [43] [50], various subtasks of diagnosis [40] [50] [1], program debugging [2], design criticism and verification [17] [20], prediction [39], automatic software- hardware reconfiguration [32], explanation [12] [30], and control of simulation [45] [31].

FR is complementary to the more common "bottom-up" device representations in which the behavioral characteristics of each component or process in isolation is represented, and the behavior of the whole is inferred, given information about how components or processes are combined. FR differs in taking a more "top-down" view in which device functions are explicitly represented, along with the roles that components and processes play in achieving those functions. FR is especially appropriate for representing a designer's intent, but is also suitable for representing unintended functions and behaviors. During this project we made progress in combining the strengths of FR and component-centered representations. Korda, Josephson, and Moberg [31, 32] demonstrated the use of an FR-based representation to direct simulation processes which are based on a VHDL-like representation. Iwasaki and Chandrasekaran [20] (included in the Appendix) demonstrated a way to do design verification, in a device-modeling environment called DME [21], by using FR in conjunction with qualitative simulation.

During this project we enhanced the representations for device functions, device structure, and causal processes; and devised and investigated reasoning strategies able to use these representations to accomplish various problem-solving subtasks supportive of design, diagnosis, and redesign.

FR recognizes the following major datatypes (classes and subclasses):

DEVICES - Components of devices are devices in their own right. A device may have associated PORTS.

A device will usually have a description of its structure - given as a set of components, their associated ports, and a set of CONNECTIONS between ports. Connections allow the passage of SUBSTANCES, which may be material (e.g., water), or abstract (e.g., heat or information). Ports and connections come in types according to the types of the substances they will pass. Device structure may also have other kinds of description (e.g., spatial, not just connection topology), but this aspect has not been very much explored in the context of FR. In principle a device may have separate descriptions depending on what perspective is taken; for example a wire may be viewed as a perfect conductor for some purposes, to have a constant but low resistance for other purposes, or to have a resistance which varies with temperature for others. While we recognized the need for multiple perspectives, we will probably not incorporate it into our representations for some time, pending a better theory of when perspective switching is appropriate.

STATES - A "state" is characterized by a partial description of the device or its environment at a moment or over an interval of time. This is usually given as a Boolean combination of predicates over STATE-VARIABLES (which may be discrete or continuous). A device has MODES which are states of the device. For example a valve may be Fully-open, Partially-open, or Closed. Every device has two major modes: NORMAL, and ABNORMAL. Known MALFUNCTION MODES are represented as sub-states of the Abnormal mode. In general Abnormal means that device behavior cannot be predicted according to the specifications of the Normal mode, whether or not it can be predicted as belonging to some Malfunction state for which a representation has been given.

FUNCTIONS - A device has a set of functions associated with each of its modes. For a Normal mode the functions may be marked as INTENDED FUNCTIONS or SIDE EFFECTS. Keuneke (1989,1991) distinguished four types of functions: to ACHIEVE, to PREVENT, to MAINTAIN, and to CONTROL. Iwasaki and Chandrasekaran (1992) have added ALLOW. Each function type is specified somewhat differently. We will focus here on Achieve functions, since the representation for them is the oldest and best developed.

An ACHIEVE function is associated with an ordered pair of states, called the IF STATE and the TO-MAKE STATE. The idea is that the To-Make state is achieved, starting from the If state, by using the particular function of the device. Often the To-Make state is described by its difference from the

If state. To explain how the To-Make state is achieved, one points to the responsible device and function. The If and To-Make states may refer to conditions at one or more ports; thus the activity of transforming values at input ports to values at output ports is a kind of Achieve function. This allows FR to absorb structure-behavior representations (such as VHDL, but there are many others) that are based only on components, ports, connections, and mathematical transformations from inputs to outputs. Sometimes an additional state is associated with a function called the PROVIDED STATE. It is used to specify conditions (other than those specified by the If state) under which the function can be expected to achieve its To-Make state, e.g., standard operating conditions. One reason for representing the Provided state is so that, if it is detected that the To-Make state has not been achieved, even though the If state did occur, then a diagnostic process will know to check whether the Provided state obtained, rather than simply concluding that the device had malfunctioned. Sometimes a TRIGGERING STATE is also associated with a function, giving conditions under which the device is expected to immediately begin a process to achieve the function.

CAUSAL-PROCESS DESCRIPTIONS - Functions are accomplished by way of causal processes. If the process is known for a particular function, a Causal Process Description (**CPD**) is associated with the representation for the function. The CPD is a "causal story" describing how the function is accomplished. A CPD is a directed graph, where the nodes are States and the edges are Annotated State Transitions. The CPD may be cyclic, but must include a directed path from the If state to the To-Make state (for Achieve functions).

ANNOTATED STATE-TRANSITIONS - An annotated state transition may be used to represent an actual change in the represented device, or merely a convenient change in state description. In the former case a causal explanation would be appropriate, but in the latter case an inferential explanation. For example the transition:

Temperature (location-1, 150° C)

Temperature (location-1, abnormally-high)

might be explained as an abstraction step, which would depend on knowledge of normality conditions.

An annotation associates a data object with the state transition; the data object may be of any of four types:

- CPD - The process by which the state transition occurs is described in more detail.
- INFERENCE - A change of state description, rather than an actual change of underlying state. The inference shows how the second state description follows from the first one, given the requisite knowledge.
- FUNCTION of a Component - A particular component, by performing one of its functions, is responsible for the state transition. FR thus provides a kind of recursive decomposition: functions are explained by the causal processes by which they are achieved, and causal processes are explained by the functions of the components that are responsible for state transitions that make up the causal process. This is the way the FR represents how the functions of a device arise from the functions of the device's components.
- GENERAL KNOWLEDGE PRINCIPLE - The state transition can be understood to occur as the result of some general principle, for example falling as a result of gravity, or increasing in temperature due to friction. A convenient way to represent general knowledge principles, especially scientific laws, is in the form of mathematical equations or functions relating parameters of the antecedent state to parameters of the subsequent one. When this is done, the FR can be used to guide numerical simulations: the FR is used to organize the set of equations constituting the mathematical model of the system, and then the representation guides the propagation of numerical values through the set of equations. This way the FR representation qualitatively organizes the causal dependencies, with the numerical equations giving the precise details and the basis for calculations. This use of FR has been explored extensively by Sticklen and his colleagues at Michigan State.

A state transition is "what is accomplished," and its annotation explains "how it is accomplished." Thus the transition represents a kind of *role* (the need to get from the one state to the other), and the annotation points to the *role filler*. More than one annotation may be given, corresponding to knowledge of more than one "actor" capable of playing the role. (This way FR can encode "multiple realizability.") Besides its annotations, a state transition may have an associated TRANSITION CONDITION (a state) under which the transition can be expected to occur. Such a condition can be tested during a simulation to determine whether the state variables should be updated (e.g., whether associated equations apply at that point).

Multiple state transitions are allowed to, and from, the nodes in a CPD. By associating conditions with the state transitions, OR and AND branching can be represented, and used to represent alternative and concurrent causal pathways.

As we said, a CPD (causal-process description) is a directed graph whose edges are annotated state transitions. Inference procedures may traverse CPDs in either direction, forwards or backwards. Traversing in the forwards direction, "consequence finding," moves from cause to effect, and supports *predictive inference*. Traversing backwards, "antecedent finding," moves from effect to cause, and supports *abductive inference*. *Design* and *planning* are logically dependent on prediction. *Diagnosis* and *process monitoring* (any that goes beyond directly-observables) are logically dependent on abduction. [By saying "logically dependent" we emphasize that the prediction and abduction do not necessarily happen explicitly at run time.] Since a device represented in FR can be considered an organized assembly of CPDs, each of which is an organized assembly of annotated state transitions, the device representation as a whole supports prediction and abduction, and so it supports design and process monitoring.

FR can be used to represent any "mechanism," not just human-designed artifacts. All that is required is that it be possible to take a "functional view," or "design stance," towards the mechanism. For example Sticklen (1987) used FR to represent portions of the human immune system, and more recently to represent portions of an ecosystem. One could use FR to represent the causal processes by which clouds achieve their "function" of producing rain. The mission of FR is to represent "how it works to do such and so," not per se on representing a designer's intended functions, though this is an appropriate use. The distinctive contributions of FR for giving (part of) the rationale for designs is described in "Functional Representation as Design Rationale," [13]. A reprint of this paper is included as an appendix to this report.

A mechanism represented in FR need not be physical; for example Allemang (1990) has used it for representing computer programs. To the degree that they are understood, FR can be used to represent the mechanisms that drive an economy towards inflation, and those that polarize a society towards civil war.

Allemang showed how program debugging can be guided by an FR representation of a program's intended functionality and method. Goel (1989) showed how to use FR representations for organizing case libraries of designs, and how the cases can be indexed by function, so a designer can retrieve candidate designs for components or whole devices. Weintraub

(1991) used FR to represent how a particular knowledge-based diagnostic system works, and showed how to use the representation for knowledge-base refinement using explanation-based learning from diagnostic mistakes. Chandrasekaran, Goel, and Iwasaki (1993) describe the use of FR for capturing design rationale. Darden, Moberg and Josephson (1990,1992) used FR to represent a historical scientific theory in the domain of transmission genetics, and used it to model scientific theory change as "anomaly-driven redesign," i.e., as diagnosing and fixing a fault in the theory. Sticklen and his colleagues have been using FR for organizing simulations. Iwasaki and Chandrasekaran (1992) demonstrated the use of FR for design verification. FR representations have been used for diagnosis. Sticklen (1987) and Punch (1989) explored combining FR-based diagnosis with "compiled" diagnosis. Keuneke (1989) showed how to use a functional representation to verify diagnostic hypotheses by constructing causal stories. DeJongh (1991) used an FR representation of antibody-antigen reactions in a SOAR-based system for medical test interpretation; his system used SOAR's learning mechanism (chunking) to incrementally "compile" knowledge for hypothesization, from causality represented in FR.

Nothing in an FR-based representation constrains the causality to be correct. One could represent the causal mechanisms posited by the Aristotelian theory of gravitation, or by a shaman's demonology. While correctness of an FR representation cannot be guaranteed from within the representation, certain kinds of consistency can be enforced, e.g., that a CPD, if it operates as described, will indeed achieve the state required by a function. FR is intended to capture key aspects of the "logic of comprehension," more specifically, important conceptual connections that make up the core of "causal understanding." FR is not complete theory. Yet it has already displayed impressive functionality and versatility, and the main path to its improvement should be empirical: by applying it and criticizing the results.

Visual Reasoning

It is apparent that engineering descriptions of devices and processes have often been expressed in the forms of diagrams and schematic pictures. Diagrammatic and pictorial representations clearly play important roles in human problem solving, as has been noticed by philosophers, cognitive psychologists, design theorists, logicians and AI researchers. Philosophers have been interested in the nature of mental imagery for a very long time,

and debates about the reality and nature of mental images and visual representations have also raged in Psychology. Design theorists have always been interested in the role of sketches and diagrams as design aids. Modern formal logicians have treated diagrams as merely "heuristic" aids to be discarded once the correct path to the real proof is obtained, but historically logicians often made use of diagrams of various sorts in conveying and using logic. AI, while it flirted with diagrams in its early decades, especially in the early work on geometric theorem proving, has neglected, until quite recently, focused research on the possible advantages of such representations. In AI there is a renewed interest in integrating symbolic and perceptual representations[1].

Diagrams preserve locality information, or represent it directly. Several visual "predicates" seem to be efficiently computed by the visual architecture from this sort of information, e.g., neighborhood relations, relative size, intersections, and so on. This ability makes certain types of inferences easy or relatively direct. (It should be emphasized that only some visual predicates are particularly easily computed by the visual architecture. There exist numerous inferences for which there is information directly available in the diagram, but which the visual system is not necessarily good at making. For example, given a large circle and a smaller circle, the visual system can directly tell that one is smaller than the other, but given two complicated shapes, where one of the shapes has a smaller area than the other, the visual architecture cannot compare them directly or easily without additional measurements and calculation.) This ability to make some visual inferences directly explains the role of diagrams in problems which are essentially spatial, such as geometry problems. Even here there are interesting strategies by which the diagram can be additionally manipulated. Additional constructions can be made on the diagram, enabling a new set of visual inferences to be made in the next cycles of reasoning. Also, symbolic annotations are made on the diagram which enables a new round of inferences to be made, not by the visual architecture but by use of information in the conceptual modality. This inference may set up information which may then enable additional visual inferences. This extremely interlaced sequence of visual and symbolic inference-making is what gives this whole approach its power: each modality makes the inferences it is best suited for, and then sets up additional information which makes it possible for the other modality to make another set of inferences for which it is best suited.

---

[1]The AAAI Spring Symposium, Reasoning With Diagrammatic Representations, was held at Stanford on March 25-27, 1992. The Symposium was organized by B. Chandrasekaran, Yumi Iwasaki, Hari Narayanan, and Herbert Simon.

In this project we have investigated the role of images and diagrams in qualitative reasoning about the physical world. Narayanan and Chandrasekaran (included in the Appendix to this report) illustrate how much of our commonsense knowledge about how objects in the world behave under various forces and collisions is actually in the form of abstract perceptual chunks that directed the reasoning activity. That is, large parts of both the knowledge and reasoning are in the visual domain. Representation of object configuration diagrams and predictive knowledge indexed by shapes as well as visual events were emphasized as central problems. This is to be contrasted with much of the current work in qualitative physics which emphasizes symbolic and axiomatic reasoning in this task.

Reconfigurable Devices and Other Applications

Avionics Design: Issues, Methods, and Results

During the third year of the project, we tested our representations on the challenges of reconfigurable devices for aerospace domains. Some devices are designed so that they can be reconfigured to adapt to events and changing conditions. Reconfigurability complicates reasoning about devices, and makes additional demands on device representations that would support such reasoning. A device may have variable potential functionality (not all of which will be simultaneously realized). Some mappings between functions and the causal processes used to attain them are flexible, that is, device functions will be achieved by different causal processes, depending on configuration and circumstances. We have arrived at a treatment for reconfigurable devices which is at least able to handle the reconfigurability present in the range of execution options available for realizing specific aircraft mission and tactical goals. We have built a prototype system that demonstrates how both quantitative and qualitative forms of simulation may be supported within a unified model of a device. (This is described in "Representing function for reasoning about software-hardware reconfiguration," included in the Appendix to this report.) In this system, design time reconfigurability tradeoffs on capacities needed for hardware devices can be explored using different methods of prediction.

Specific results on problem solving and multi-level causal modeling for the aerospace and avionics domain have been reported in the recent 1992 annual report, and will here only be briefly reviewed. Two technical

reports are included in the Appendix to this report: "Representing function for reasoning about software-hardware reconfiguration," and "Multilevel Causal-Process Modeling: Bridging the Plan, Execution, and Device-Implementation Gaps."

Designing aerospace systems can potentially benefit significantly from multilevel modeling that relates higher level mission or tactical functions to the implementing levels of supporting avionics and aircraft system subfunctionality. A particular benefit will be in improved levels of fidelity of test and simulation for loads on underlying devices. Such improvements can lead to architectural and hardware support for load balancing strategies discovered while in the design critique and verification process. A model that relates high level functions and requirements to lower level systems is needed for an integrated redesign system that can track and guide design tests and modifications. [The redesign task, of course, includes many subordinate tasks that are spawned as the task is pursued; simulation, abduction, classification and diagnosis can become appropriate during redesign.]

Also, because the computational demands of simulation can easily grow beyond capabilities that can reasonably be expected to exist in the next few decades, we have begun to explore prototype environments for investigating the feasibility and value of merging various methods of simulation and prediction. At present a prototype has shown how a simulation system that aids hardware design can make use of a mixture of methods. The system has investigated the capacities of that hardware in relation to constraints arising from the software functionality that the hardware is to support.

We selected aerospace and avionics as a domain to test recent advances in causal process description and functional decomposition. In previous and concurrent work, the domains of process engineering, simple manufactured devices, and biological processes have been found to be useful application areas for LAIR causal modeling techniques. One reason for varying domains is to ensure that the techniques have wide applicability. A second is that it is thereby likely that we will arrive at a self-contained and useful set of domain-independent primitive constructs, with well-defined modes of composition, specification, and instantiation. The aerospace and avionics domain is representative of emergent technology domains in which information and control systems, and human agents, interact in complex ways to achieve useful functions. Such systems are difficult to design and verify, and because of their economic or political importance, deserve to benefit from advances in knowledge based system technology.

The availability of VHDL libraries of avionics devices has also contributed to selecting avionics and aerospace for attention. Trends in knowledge based systems favor investigation of ways to ameliorate the burden of knowledge acquisition by opportunistically drawing upon the information that has been and is being accumulated in some computationally accessible forms. It would be clearly helpful to knowledge based problem solving technology if the effort is made to find more ways to draw upon conventionally available data and resources. During the final year of the project we have investigated several points of application of model-based reasoning technology in the domain of the new generation of avionics systems, especially the models that involve explicit hierarchical representation of function and structure.

Multilevel Causal Process Models

The experience with the Pilot Associate (PA) project shows that it is important to consider how a future knowledge base might support knowledge compilation for distributed PA modules. Maintenance of a technical knowledge base has been indirectly studied in the Learning System for Pilot Aiding (LSPA) project. Using explanation based generalization techniques, the LSPA project has shown how new plans can be included in a knowledge base after the plans are learned from a record of pilot responses in a simulated mission. Advanced Avionics and the Role of VHDL models: In a future technical knowledge base, it will be important to be able to align top level requirements with the lowest level performance measurements on equipment, including avionics subsystems at various granularities. VHDL models of avionics subsystems can be used to explore new capabilities of devices when the new capabilities affect mission critical performance parameters. This exploration can occur even in advance of the actual production of the devices. But to do so will require extensive modeling of the dependencies and interdependencies as they span both abstraction levels and subsystems. We have been concerned with developing modeling techniques expressive enough to capture these dependencies, and to make them available for a wide range of types of problem solving.

Multilevel Functional Decompositions

During the third year of research, we developed functional decompositions and causal process models for specific real world problems that would

challenge our modeling techniques. One goal was to develop insight into what organizational principles would be needed for device modeling in the aerospace engineering domain that could support design, troubleshooting, tutorial, and simulation uses. One objective was to develop a representation (model) that takes a top level function (a mission function with temporal features to it and a time course during which different subfunctions are scheduled) and show a vertical slice of how it gets decomposed into subfunctions and ultimately into hardware components.

Much of our thinking about ordinary devices and how they work involves viewing those objects as contributing to various causal processes that make the device realize its functions. Causation has been called "the cement of the universe," and few would contest the centrality of causal relations, and the role they play in organizing our information about systems, events, components, processes, states, subsystems and functions. One organization of our causal knowledge of a system, especially a complex system such as a pilot flying an aircraft, is from overall functionality down to the subsystems with their subcomponents and their contributing functionalities. A functional decomposition of aircraft flights is primarily of value to show how to move from higher levels of functional description to lower levels of causal processes that realize functions of interest. A decomposition in which the functional parts are both represented and related then provides the "integrative glue" for subsequent problem solving across the levels. If we want to know how an equipment change, for example, might effect parameters governing a maneuver for accomplishing some flight goal, the dependencies marked in a functional decomposition can show us what to look for, and how to figure out what has changed. Results of using FR for a multi-level functional representation connecting maneuver-level phenomena with hardware-level phenomena are reported in a paper that is included in the Appendix of this report.

Use of Functional Models for Controlling Simulation for Hardware Estimation

We have made considerable progress in developing tools and techniques for carrying out "smarter" simulations of software and hardware avionics subsystems. Computer simulation of devices usually consumes significant amount of computational time. Making simulation more efficient and goal-oriented will help to overcome this problem. One objective of the work reported here is to show how simulations may be made more efficient by blending functional representation (FR) with structure-based models.

Our system, aimed at intelligent control of simulation, consists of two main layers:

1. The upper layer generates a set of higher simulation goals based on the high level tasks of the problem-solver. Some subtasks of diagnosis, design support, possibly combined with the representation of the role of the device within a more complex device, can provide high level goals for simulation.

2. The lower layer economically achieves the basic simulation goals generated by the upper layer. The savings on simulation time are gained by focusing on simulation goals while making use of the capabilities of FR to focus on the relevant causal pathways to project.

The objectives of our investigations of task based control of simulation have been:

- to show how FR can be smoothly combined with structure-based models for simulation in a common framework.
- to determine the advantages of combining the two, and whether FR can help control the simulation.
- to determine whether higher level goals of the problem-solver can be used to create lower level goals for the simulator.
- to investigate the tradeoffs between controlling simulation by FR and simulating brute-force by using the structural model alone.

Two prototype systems were developed to accomplish these objectives. The initial results have shown in principle that simulation of large systems can be done efficiently by meshing less computationally costly techniques with focused invocation of relevant computationally demanding methods. Details of the preliminary results are found in a paper included in the appendix to this report.

References

Missing numbers can be found in the following section, which lists papers supported by the project.

[3] D. C. Brown and B. Chandrasekaran. Knowledge and control for a mechanical design expert system. IEEE Computer Magazine, pages 92--100, July 1986.

[4] David C. Brown and B. Chandrasekaran. Design Problem Solving: Knowledge Structures and Control Strategies. Morgan Kaufmann, San Mateo, CA, 1989.

[6] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. IEEE Expert, 1(3):23--30, 1986.

[7] B. Chandrasekaran. Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. Knowledge Engineering Review, 3(3):183--210, September 1988. Issue 3.

[12] B. Chandrasekaran and J. R. Josephson. Explanation, problem solving, and new generation tools: A progress report. In Proceedings of the Expert Systems Workshop, pages 122--126, Pacific Grove, CA, April 1986. Science Applications International Corporation.

[16] Matthew DeJongh. Causal Processes in the Problem Space Computational Model: Integrating Multiple Representations of Causal Processes in Abductive Problem Solving. PhD thesis, The Ohio State University, Columbus, OH, 1991.

[18] Ashok Goel, P. Sadayappan, and John R. Josephson. Concurrent synthesis of composite explanatory hypotheses. In Proceedings of the Seventeenth International Conference on Parallel Processing, volume III, pages 156--160, St. Charles, IL, August 1988. The Pennsylvania State University Press.

[21] Y. Iwasaki and C. M. Low. Device modeling environment: An integrated model-formulation and simulation environment for continuous and discrete phenomena. In Proceedings of the First International Conference on Intelligent Systems Engineering, pages 141--146. The Institution of Electrical Engineers, 1992.

[22] T. Johnson, B. Chandrasekaran, and J. W. Smith Jr. Generic tasks and SOAR. In Working notes of the AAAI Spring Symposium on Knowledge System Development Tools and Languages, Menlo Park, CA, 1989. AAAI.

[24] Todd R. Johnson and Jack W. Smith. A framework for opportunistic abductive strategies. In Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society, pages 760--764, Hillsdale, NJ, August 1991. Lawrence Erlbaum Associates. Chicago.

[25] John R. Josephson. Reducing uncertainty by using explanatory relationships. In Proceedings of the Space Operations Automation and Robotics 1988 Workshop, pages 149--151, Dayton, OH, 1988. NASA, USAF and Wright State University.

[26] John R. Josephson, B. Chandrasekaran, and Jack W. Smith, Jr. Assembling the best explanation. In Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems, pages 185--190, Denver, CO, December 1984. IEEE Computer Society. A revised version by the same title is available as a LAIR technical report.

[27] John R. Josephson, B. Chandrasekaran, Jack W. Smith, and Michael C. Tanner. A mechanism for forming composite explanatory hypotheses. Technical report, OSU CIS LAIR, 1987. This paper is a revised and expanded version of an earlier paper by the same title which appeared in IEEE Transactions on Systems, Man and Cybernetics, Special Issue on Casual and Strategic Aspects of Diagnostic Reasoning, 1986.

[28] John R. Josephson, Diana Smetters, Richard Fox, Dan Oblinger, Arun Welch, and Gayle Northrup. The integrated generic task toolset, Fafner release 1.0., Technical report, The Ohio State University, Laboratory for Artificial Intelligence Research, Columbus, OH, 1989.

[29] A. Keuneke. Device representation: The significance of functional knowledge. IEEE Expert, April 1991.

[30] Anne M. Keuneke. Understanding Devices: Causal Explanation of Diagnostic Conclusions. PhD thesis, The Ohio State University, Columbus, OH, 1989.

[33] S. Mittal, B. Chandrasekaran, and J. Sticklen. PATREC: A knowledge-directed data base for a diagnostic expert system. IEEE Computer Special Issue, 17:51--58, September 1984.

[35] Dale Moberg and John R. Josephson. Implementation note on diagnosing and fixing faults in theories. In Jeff Shrager and Pat Langley, editors, Computational Models of Scientific Discovery and Theory Formation, pages 347--353. Morgan Kaufmann, San Mateo, CA, 1990.

[39] M. Pegah, W. E. Bond, and J. Sticklen. Representing and reasoning about the fuel system of the McDonnell Douglas F/A-18 from a functional perspective. IEEE Expert, 1992.

[40] William F. Punch III. A Diagnosis System Using a Task Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning. PhD thesis, The Ohio State University, Columbus, OH, 1989.

[41] William F. Punch III, M. C. Tanner, and J. R. Josephson. Design considerations for PEIRCE, a high-level language for hypothesis assembly. In Kamal N. Karna, Kamran Parsaye, and Barry G. Silverman, editors, Proceedings of the Expert Systems in Government Symposium, pages 279--281, McLean, VA, October 1986. IEEE Computer Society Press.

[43] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem solving systems. In J. L. Kolodner and C. K. Riesbeck, editors, Experience, Memory and Reasoning, pages 47--73. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[44] J. Sticklen, B. Chandrasekaran, and W. Bond. Distributed causal reasoning, Knowledge Acquisition, 1:139--162, 1989.

[45] J. Sticklen, A. Kamel, and W. E. Bond. Integrating quantitative and qualitative computations in a functional framework. Engineering Applications of Artificial Intelligence, 4(1):1--10, 1991.

[46] J. Sticklen, A. Kamel, and W. E. Bond. A model-based approach for organizing quantitative computations. In Second Annual Conference on AI Simulation and Planning in High Autonomy Systems, Orlando, FL, 1991.

[47] J. Sticklen, A. Kamel, and W. E. Bond. A model-based approach for organizing quantitative computations. In Second Annual Conference on AI, Simulation and Planning in High Autonomy Systems, Orlando, FL, 1991.

[48] J. Sticklen, A. Kamel, M. Hawley, and V. Adegbite. Fabricating composite materials: A comprehensive problem solving architecture based on a generic task viewpoint. IEEE Expert, March 1992.

[49] J. Sticklen and R Tufankji. Utilizing a functional approach for modeling biclogical systems. Mathematical and Computer Modeling, 16:145--160, 1992.

[50] Jon Sticklen. MDX-2: An Integrated Medical Diagnostic System. PhD thesis, The Ohio State University, Columbus, OH, 1987.

[51] M. A. Weintraub. An Explanation-Based Approach for Assigning Credit. PhD thesis, The Ohio State University, Columbus, OH, 1991.

Papers and Publications Supported by the Project

The following is a list of papers and publications that were supported in some way by the project reported here. This list is certainly not complete.

[1] Prabal K. Acharyya. Causality and knowledge-based diagnosis of nuclear power plants. Master's thesis, The Ohio State University, Columbus, Ohio, 1992.

[2] D. Allemang. Understanding Programs as Devices. PhD thesis, Ohio State University, 1990.

[5] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson. The computational complexity of abduction. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, Knowledge Representation, pages 25--60. MIT Press, Cambridge, MA, 1992. Also appears in volume 49 of Artificial Intelligence.

[8] B. Chandrasekaran. Task structures, knowledge acquisition, and learning, Machine Learning, 4:339--345, 1989.

[9] B. Chandrasekaran. Design problem solving: A task analysis. Artificial Intelligence Magazine, 11(4):59--71, 1990.

[10] B. Chandrasekaran. Models vs. rules, deep vs. compiled, content vs. form: Some distinctions in knowledge systems research. IEEE Expert, 1991.

[11] B. Chandrasekaran and Todd R. Johnson. Generic tasks and task structures: History, critique and new direction. In J. M. David, J. P. Krivine, and R. Simmons, editors, Second Generation Expert Systems. Springer-Verlag, New York, NY, to appear.

[13] B. Chandrasekaran, Goel Ashok K., and Yumi Iwasaki. Functional representation as design rationale. IEEE Computer, January 1993. Special Issue on Concurrent Engineering.

[14] Lindley Darden, Dale Moberg, Satish Nagarajan, and John Josephson. Anomaly driven redesign of a scientific theory: The Transgene.2 experiments. Technical Report 91-LD-TRANSGENE, Laboratory for

Artificial Intelligence Research, The Ohio State University, Columbus, Ohio, 1991.

[15] Lindley Darden, Dale Moberg, Sunil Thadani, and John Josephson. A computational approach to scientific theory revision: The TRANSGENE experiments. Technical report, The Ohio State University, Laboratory for Artificial Intelligence Research, Columbus, OH, 1992. Ordering code: 92-LD-TRANSGENE.

[17] A. K. Goel. Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving. PhD thesis, The Ohio State University, 1989.

[19] David H. Herman. An extensible, task-specific shell for routine design problem solving. PhD thesis, The Ohio State University, Columbus, Ohio, 1992.

[20] Y. Iwasaki and B. Chandrasekaran. Design verification through function- and behavior-oriented representations: Bridging the gap between function and behavior. In J. S. Gero, editor, Artificial Intelligence in Design, pages 597--616. Kluwer Academic Publishers, Netherlands, 1992.

[23] T. R. Johnson. Generic Tasks in the Problem-Space Paradigm: Building Flexible Knowledge Systems while using Task-Level Constraints. PhD thesis, The Ohio State University, Columbus, OH, 1991.

[31] Susan T. Korda, John Josephson, and Dale Moberg, Combining functional representation and structure-based models for smarter simulation. Technical report, The Ohio State University, Laboratory for Artificial Intelligence Research, 1992.

[32] Susan T. Korda, John Josephson, and Dale Moberg. Representing function for reasoning about software-hardware reconfiguration. Technical report, The Ohio State University, Laboratory for Artificial Intelligence Research, 1992.

[34] Dale Moberg, Lindley Darden, and John Josephson, Representing and reasoning about a scientific theory, in AAAI Workshop on Communicating Scientific and Technical Knowledge, pages 58--64, San Jose, CA, 1992. AAAI. July 12-16, San Jose Convention Center.

[36] N. H. Narayanan and B. Chandrasekaran. Qualitative simulation of spatial mechanisms: a preliminary report. In Proc. 21st Annual Conference on Modeling and Simulation, Pittsburgh, PA, 1990.

[37] N. H. Narayanan and B. Chandrasekaran. Rigid body motion prediction by analogical simulation. In Proc. AAAI-90 Workshop on AI and Simulation, pages 69--74, Boston, MA, 1990.

[38] N. H. Narayanan and B. Chandrasekaran. A visual approach to qualitative kinematics. In Proc. AAAI-90 Workshop on Qualitative Vision,, pages 72--76, 1990.

[42] William F. Punch III, M. C. Tanner, J. R. Josephson, and J. W. Smith. PEIRCE: A tool for experimenting with abduction. IEEE Expert, 5(5):34--44, October 1990.

[52] B. Chandrasekaran, "QP is More Than SPQR and Dynamical Systems Theory: Response to Sacks and Doyle," to appear in Computational Intelligence.

[53] B. Chandrasekaran and N. Hari Narayanan, "Perceptual Representation and Reasoning," appears in *Intelligent Systems*, edited by L. S. Sterling, Plenum Press, New York, 1993.

[54] John and Susan Josephson, Ed., Abductive Inference: Computation, Philosophy, Technology, Cambridge University Press (forthcoming).

[55] N. H. Narayanan, *Imagery, Diagrams, and Reasoning*, Ph.D. thesis, The Ohio State University, 1992.

[56] B. Chandrasekaran, "Design problem solving: A task analysis," *Artificial Intelligence Magazine*, 11(4):59--71, 1990.

[57] B. Chandrasekaran, Todd R. Johnson, Jack W. Smith, "Task-Structure Analysis for Knowledge Modeling," *Communications of the Association for Computing Machinery*, September, 1992, pp. 124-137.

[58] B. Chandrasekaran, John R. Josephson, and Susan G. Josephson, "Form and Content Issues in the Abductive Framework for Recognition, LAIR Technical report.

[59] Michael C. Tanner, Anne M. Keuneke, and B. Chandrasekaran, "Explanation Using Task Structure and Domain Function Models," to appear

in *Second Generation Expert Systems*, edited by Jean-Marc David, Jean-Paul Krivine, and Reid Simmons, Springer-Verlag.

[60] B. Chandrasekaran, Ashok Goel, and Yumi Iwasaki, "Functional Representation as Design Rationale," appears in *IEEE Computer, Special Issue on Concurrent Engineering*, 1993.

[61] B. Chandrasekaran and William Swartout, "Explanation in Knowledge Systems: The Role of Explicit Representation of Design Knowledge," *IEEE Expert*, June 1991.

[62] Y. Iwasaki and B. Chandrasekaran, "Design verification through function- and behavior-oriented representations: Bridging the gap between function and behavior," in J. S. Gero, editor, *Artificial Intelligence in Design*, pages 597--616. Kluwer Academic Publishers, Netherlands, 1992.

[63] Marcos Vescove, Yumi Iwasaki, Richard Fikes, and B. Chandrasekaran, "CFRL: A Language for Specifying the Causal Functionality of Engineered Devices," AAAI-93.

[64] Yumi Iwasaki, Richard Fikes, Marcos Vescovi, B. Chandrasekaran, "How things are *Intended* to work: Capturing Functional Knowledge in device Design," *IJCAI*, 1993.

[65] B. Chandrasekaran and Susan Josephson, "Architecture of intelligence: The problems and current approaches to solutions," appears in *Current Science*, Vol. 64, No. 6, March 1993.

[66] B. Chandrasekaran and N. Hari Narayanan, "Perceptual Representation and Reasoning," appears in *Intelligent Systems*, edited by L. S. Sterling, Plenum Press, New York, 1993.

[67] N. Hari Narayanan and B. Chandrasekaran, "Reasoning Visually About Spatial Interactions," IJCAI-91, Sydney, Australia, Morgan-Kaufmann Publishers.

[68] Susan T. Korda, John R. Josephson, and Dale Moberg, "Combining Functional Representation and Structure-Based Models for Smarter Simulation," LAIR technical report.

[69] Susan T. Korda, John R. Josephson, and Dale Moberg, "Representing function for reasoning about software-hardware reconfiguration,"

Technical report, The Ohio State University, Laboratory for Artificial Intelligence Research, 1992.

[70] Keith Levi, Dale Moberg, Christopher Miller, Fred Rose, "Multilevel Causal-Process Modeling: Bridging the Plan, Execution, and Device-Implementation Gaps."

Appendices: Included Papers

B. Chandrasekaran, "Design problem solving: A task analysis," *Artificial Intelligence Magazine*, 11(4):59--71, 1990.

B. Chandrasekaran, Todd R. Johnson, Jack W. Smith, "Task-Structure Analysis for Knowledge Modeling," *Communications of the Association for Computing Machinery*, September, 1992, pp. 124-137.

B. Chandrasekaran, John R. Josephson, and Susan G. Josephson, "Form and Content Issues in the Abductive Framework for Recognition, LAIR Technical report.

Michael C. Tanner, Anne M. Keuneke, and B. Chandrasekaran, "Explanation Using Task Structure and Domain Function Models," to appear in *Second Generation Expert Systems*, edited by Jean-Marc David, Jean-Paul Krivine, and Reid Simmons, Springer-Verlag.

B. Chandrasekaran and William Swartout, "Explanation in Knowledge Systems: The Role of Explicit Representation of Design Knowledge," *IEEE Expert*, June 1991.

B. Chandrasekaran, Ashok Goel, and Yumi Iwasaki, "Functional Representation as Design Rationale," appears in *IEEE Computer, Special Issue on Concurrent Engineering*, 1993.

Y. Iwasaki and B. Chandrasekaran, "Design verification through function- and behavior-oriented representations: Bridging the gap between function and behavior," in J. S. Gero, editor, *Artificial Intelligence in Design*, pages 597--616. Kluwer Academic Publishers, Netherlands, 1992.

Marcos Vescove, Yumi Iwasaki, Richard Fikes, and B. Chandrasekaran, "CFRL: A Language for Specifying the Causal Functionality of Engineered Devices," AAAI-93.

Yumi Iwasaki, Richard Fikes, Marcos Vescovi, B. Chandrasekaran, "How things are *Intended* to work: Capturing Functional Knowledge in device Design," *IJCAI*, 1993.

B. Chandrasekaran, "QP is More Than SPQR and Dynamical Systems Theory: Response to Sacks and Doyle," to appear in *Computational Intelligence.*

Susan T. Korda, John R. Josephson, and Dale Moberg, "Combining Functional Representation and Structure-Based Models for Smarter Simulation," LAIR technical report.

Susan T. Korda, John R. Josephson, and Dale Moberg, "Representing function for reasoning about software-hardware reconfiguration," Technical report, The Ohio State University, Laboratory for Artificial Intelligence Research, 1992.

Keith Levi, Dale Moberg, Christopher Miller, Fred Rose, "Multilevel Causal-Process Modeling: Bridging the Plan, Execution, and Device-Implementation Gaps."

B. Chandrasekaran and N. Hari Narayanan, "Perceptual Representation and Reasoning," appears in *Intelligent Systems*, edited by L. S. Sterling, Plenum Press, New York, 1993.

N. Hari Narayanan and B. Chandrasekaran, "Reasoning Visually About Spatial Interactions," IJCAI-91, Sydney, Australia, Morgan-Kaufmann Publishers.

B. Chandrasekaran and Susan Josephson, "Architecture of intelligence: The problems and current approaches to solutions," appears in *Current Science*, Vol. 64, No. 6, March 1993.

2

Design Problem Solving: A Task Analysis[1]

B. CHANDRASEKARAN
Laboratory for AI Research, Department of Computer & Information Science, The Ohio State University, Columbus, OH 43210, USA

THE TASK-STRUCTURE METHODOLOGY

Design problem solving is a complex activity involving a number of subtasks, and a number of alternative methods potentially available for each subtask. The structure of tasks has been a key concern of recent research in task-oriented methodologies for knowledge-based systems (Clancey, 1985; Chandrasekaran, 1986; McDermott, 1988; Steels, 1990). One way to conduct a task analysis is to develop a *task-structure* (Chandrasekaran, 1989) that lays out the relation between a task, applicable methods for it, the knowledge requirements for the methods, and the subtasks set up by them. The major goal of this chapter is to develop a task structure for design as a knowledge-based problem-solving activity.

Design as search in a space of subassemblies

Designing artefacts that are meant to achieve some functions within some constraints is an important class of design with characteristic properties (Goel and Pirolli, 1989). We concentrate on this class of design problems in this chapter.

For sufficiently complex versions of the design problem, a common theme emerges for design as a process: it involves mappings from the

[1] This work has evolved over a number of years. Earlier versions have appeared as Chapter 2 of Brown and Chandrasekaran (1989) and in *Research in Engineering Design* (1989), 1, 75–86. This version previously appeared in *AI Magazine* (1990), 11, 59–71.

Knowledge Aided Design
ISBN 0-12-298250-9

space of design specifications to the space of devices or components (often referred to as mapping from behaviour to structure), typically conducted by means of a search or exploration in the space of possible subassemblies of components. This is in fact the origin of the frequent suggestion that design is a *synthetic* task.

The design problem is formally a search problem in a very large space for objects that satisfy multiple constraints. Only a vanishingly small number of objects in this space constitute even "satisficing", not to speak of optimal, solutions. What is needed to make design practical are strategies that radically shrink the search space.

Set against the view of design as a deliberative problem-solving process is the view of design as an "intuitive", almost instantaneous, process in which a design solution comes to the mind of the designer. Artistic creations and scientific theories are often said by their creators to have occurred to them in this manner. Even when a plausible solution occurs in this way, the proposal still needs to be evaluated, critiqued and modified by deliberatively examining alternatives. That is, except in simple cases, deliberative processes are still essential for real-world design.

Functions, constraints, components and relations

A designer is charged with specifying an artefact that delivers some functions and satisfies some constraints. For each design task, the availability of a (possibly large and generally only implicitly specified) set of *primitive components* can be assumed. The domain also specifies a repertoire of *primitive relations* or connections that are possible between components. An electronics engineer, for example, may assume the availability of transistors, capacitors, and other electrical components when he is designing a waveform generator. Primitive relations in that domain are serial and parallel connections between components.

Of course, design is in general recursive: if a certain component that was assumed to be available is in fact not available, the design of that can be undertaken in the next round. However, the vocabulary of primitive components and relations may be rather different from those for the original device.

Functions can be expressed as a state or a series of states that we want the device to achieve or avoid under specified conditions. Functions may be explicitly stated as part of problem specifications, or they may be implicit in the designer's understanding of the domain. An example of an implicit function in many engineering devices is safety: e.g. a subsystem's role may only be explained as something that prevents leakage of a potentially hazardous substance, and this function may never be stated explicitly as part of the design specification (Keuneke, 1989).

Usually, design specifications will mention, in addition to desired functionalities, a number of constraints.[2] The distinction between functions and constraints is hard to pin down formally; functions *are* constraints on the behaviour or properties of the device. It is, however, useful to distinguish functions from other constraints, since the former are the primary reason why the device is desired. Design constraints can be on the properties of the artefact (e.g. "Should not weigh more than . . ."), on the process of making the artefact from its description (manufacturability constraints), on the design process itself (e.g. "I want a design within a week"), and so on. A computationally effective process of design is to generate a candidate design based on functions and then to modify it to meet the constraints.

Definition of the design task

Consider the following definition of the design task. The design problem is specified by:

- a set of functions (explicitly stated by the design consumer as well as implicit ones defined by the domain) to be delivered by an artefact and a set of constraints to be satisfied; and
- a "technology", i.e. a repertoire of components assumed to be available and a vocabulary of relations between components.

The constraints may pertain to the design parameters themselves, to the process of making the artefact, or to the design process. The solution to the design problem consists of a complete specification of a set of components and their relations, which together describe an artefact that delivers the functions and satisfies the constraints. The solution is expected to satisfy a set of implicit criteria as well, e.g. it is not much more complex or costly than plausible alternatives (ruling out Rube Goldberg devices).

The preceding definition also captures the *domain-independent* character of design as a generic activity. Planning, programming and engineering design all share the above definition, as well as many of the subprocesses, to a significant degree. Nevertheless, there are versions of the design problem for which the above definition needs to be modified or extended. Examples are:

- At the start of the design process only a minimal statement of functions and constraints may be available, and additional ones

[2] The constraints that are described as part of the design specification ought to be distinguished from the term "constraint" that appears in description of design methods, such as "constraint-directed problem solving".

may be developed in parallel with the design process itself.

- Some design problems involve extensive trade-off studies, where a part of the design process is search for ways in which the functions or the constraints may be relaxed or otherwise modified.
- "Tinkering" is a time-honoured method of invention where the design space is being explored without any specific set of functions in mind. Functions may be identified for structural configurations that arise during exploration.
- The world of primitive objects may be very open-ended, and only implicitly specified.

The design framework that I will be presenting can be extended to cover these variations.

THE TASK-STRUCTURE

Let us say we have a problem-solving task[3] T, and let M be some *method* suggested for the task. A method can be described in terms of the *operators* it employs, the *objects* that it operates on, and any additional knowledge about how to organize operator application to satisfy the goal. At the knowledge level, the method is characterized by the knowledge the agent needs to set up and apply the method. Different methods for the same task may call for knowledge of different types.

To take a simple example, for the task of multiplying two multidigit numbers, the "logarithmic method" consists of the following series of operations: *extract the logarithm* of each of the input numbers, *add* the two logarithms, and *extract the anti-logarithm* of the sum. (The operators are italicized. Their arguments as well as the results are the objects of this method.)

Note that one does not typically include, at this level of description of the logarithmic method, specifications about how to extract the logarithm or the anti-logarithm, or how to do the addition. If the computational model does not provide these capabilities as primitives, performing these operations can be set up as *subtasks* of the method. Thus, given a method, applying any of the operators of a method can be set up as a subtask. Some of the objects a method needs may be generic to a class of problems in a domain. As an example, consider hierarchical classification using a malfunction hierarchy, a common method of diagnosis. Operations of "Establish-Hypothesis" and "Refine-Hypothesis" are applied to the hypotheses in the hierarchy. These objects are useful to solve many

[3] In this chapter, I use the terms "task" and "goal" interchangeably.

instances of the diagnostic problem in the domain. If the malfunction hypotheses are not directly available, generation of such hypotheses can be set up as subtasks. A common method for the generation of such objects is compilation from so-called "deep" knowledge. Structure–function models of the device that is being diagnosed have been proposed and used as deep models to generate malfunction hypotheses (Chandrasekaran *et al.*, 1989).

There is no finite set of mutually distinct methods for a task, since there can be numerous variants on a method. Nevertheless, the term "method" is a useful shorthand to refer to a set of related proposals about organizing computation.

Types of methods. One type of method is of particular importance in knowledge-based systems: methods which can be viewed as a problem-space search (Newell, 1980). Designer-Soar (Steier, 1989) and AIR-CYL (Brown and Chandrasekaran, 1989) are examples of design systems which explore search spaces. For example, AIR-CYL can be understood as searching in a space of parameters for the components of an air-cylinder by using design plans which propose and modify parameter values.

Another class of methods consists of algorithms which directly produce a solution without any search in a space of alternatives, e.g. producing a set of design parameters by numerically solving a set of simultaneous equations. Such algorithms are only available for so-called well-structured problems.[4] Most real-world problems are ill-structured, and the role of domain knowledge is to help set up spaces of alternatives and to help control the search in those spaces.

A task analysis of this type can be continued recursively until methods whose operators are all directly achievable (within the analysis framework) are reached. In the following task analysis for design, I will explicitly indicate as subtasks only those to which I want to draw specific attention in my discussion. Other operators may exist which require additional problem solving as well.

A TASK-STRUCTURE FOR DESIGN: THE PROPOSE–CRITIQUE–MODIFY FAMILY OF METHODS

The most common top-level family of methods for design can be characterized as *Propose–Critique–Modify* (PCM) methods. These

[4] I subscribe to the view that such algorithms are simply degenerate cases of search where the agent has sufficient knowledge to make the correct choice at each choice point. But, pragmatically speaking, it is best to think of algorithmic methods as a separate type, since implementing them does not require supporting search in general.

methods have the subtasks of Proposal of partial or complete design solutions; Verification of proposed solutions; Critiquing the proposals by identifying causes of failure, if any; and Modification of proposals to satisfy design goals. These subtasks can be combined in fairly complex ways, but the following is one straightforward way in which a PCM method may organize and combine the subtasks.

Example PCM method

Step 1. Given design goal, Propose solution. If no proposal, exit with failure.
Step 2. Verify proposal. If verified, exit with success.
Step 3. If unsuccessful, Critique proposal to identify sources of failure. If no useful criticism available, exit with failure.
Step 4. Modify proposal and return to 2.

While all of the PCM methods will need to have some way to achieve the iteration in Step 4 above, there can be numerous variants on the way the methods in this class work. For example, a solution may be proposed only for a part of the design problem, a part deemed to be crucial. This solution may then be critiqued and modified. This partial solution may generate additional constraints, leading to further design commitments. Thus subtasks can be scheduled in a fairly complex way, with subgoals from different methods alternating. It is hard to identify a separate method for each such variation. The implications of this for a design architecture are discussed in the concluding sections of the chapter.

In this chapter, most of the attention is devoted to the Proposal subtask, since most of the design knowledge, per se, is used in this subtask. Every task has a default method: one which uses compiled knowledge to get a solution without any problem solving. This method is practical only in simple cases. Because this method is potentially applicable to simple versions of all tasks, and has no interesting internal structure, I will not explicitly mention it in my discussion.

A task analysis should provide a framework within which various approaches to design can be understood. I will use selected examples of existing AI systems to illustrate the ideas, but there will be no attempt to provide a survey of all AI work on design.

Methods for proposing design choices

Design proposal methods use domain knowledge to map part or all of the specifications to partial or complete design proposals. Three groups of methods can be identified:

- Problem decomposition/solution composition. In this class of methods, domain knowledge is used to map subsets of design

specifications into a set of smaller design problems. Use of design plans is a special case of decomposition methods.
- Retrieval of cases from memory which correspond to solutions for design problems which are similar or "close" to the current problem.
- Families of methods that solve the design problem as a constraint satisfaction problem and use a variety of quantitative and qualitative optimization or constraint satisfaction techniques.

Decomposition and case-based methods help reduce the size of the search spaces, since the knowledge they use can be viewed as the compilation or chunking of earlier (individual or community) search in the design space. Conversion of a design problem into one amenable to global optimization algorithms requires substantial a priori knowledge of the structure of the design problem.

Decomposition/solution composition

I will treat this method in terms of all the features that an information processing analysis calls for: types of knowledge and information needed and the inference processes that operate on this form of knowledge.

Knowledge needed is of the form $D \rightarrow D_1, D_2, \ldots D_n$, where D is a given design problem, and D_is are "smaller" subproblems (i.e. associated with smaller search spaces than D). A number of alternative decompositions for a problem may be available, in which case a selection needs to be made, with the attendant possibility of backtracking and making another choice. Repeated applications of the decomposition knowledge produce *design hierarchies*. In well-trodden domains, effective decompositions are known and little search for decompositions needs to be conducted as part of routine design activity. For example, in automobile design, the overall decomposition has remained largely invariant over several decades.

Decomposition knowledge in design generally arises when the functional specifications can be decomposed into a set of subfunctions (Freeman and Newell, 1971). Design decomposition knowledge may come in the form of part–subpart decomposition, if a direct mapping is available between functions and components.

The following are two important subgoals of the Decomposition/Solution Composition method.

- Generating specifications for subproblems. The functional and other specifications on D need to be translated into specifications for each of the subproblems $D_1, \ldots D_n$.
- Gluing the subproblem solutions into a solution to the original design problem.

In most routine design, these subtasks are not explicit: either they are solved by compiled knowledge or the problem specification already implies a solution to these problems. In the general case, however, additional problem solving is needed.

How a Decomposition/Solution Composition method might actually organize and use the subgoals is given by the following example.

Example problem decomposition/solution recomposition method

Step 1. (Search in the space of decompositions.) Choose from among alternative decompositions for the given design problem D.

Step 2. Generate specifications for subproblems in the chosen decomposition.

Step 3. Set up each subproblem as a design problem. Solve them in some order determined by control strategies and other domain knowledge (e.g. progressive deepening).

Step 4. If subproblems solved, Recompose solutions of subproblems into solutions for D, and exit.

Step 5. If failure in Steps 3 or 4, go to Step 1 to make another choice, or relax specifications and go to Step 2.

All the caveats mentioned in connection with the PCM method earlier apply. Specifically, control of how subproblems are solved may be quite variable and more complex than indicated above. Some of the sources of this complexity are discussed below.

Given a design problem, it may not always be possible to generate all the constraints for its subproblems from the original problem's specifications alone. In many domains, constraint generation for some subproblems alternates with partial design of others, which in turn provides additional information for constraints for yet other subproblems. There may be a complex process of commitments and backtracking. In extreme cases, most of design problem solving may consist of search for parameters that make all the subproblems solvable. For example, the *Propose and Revise* method (Marcus *et al.*, 1985) involves making commitments to some subparts of the design problem (Propose part) and then Revising these when some constraint for other parts of the problem are violated.

In configuration tasks (Mittal and Frayman, 1989), subproblem solutions are given as part of the problem (i.e. the desired functions are mapped into a set of key components), and the remaining task is dominated by the subtasks of specification generation and solution recomposition. In order for components A and B to be connected, certain pre- and post-conditions may need to be satisfied. If these conditions are not available a priori, they need to be derived from configuration behaviours. Discovery of connection conditions and checking of whether specific configuration proposals result in desired functional behaviours can often use simulation as a problem solving method (e.g. Kelly and Steinberg, 1982).

There can be complex dependencies between constraints among subproblems. In situations where not only are commitments for D_1 going to constrain the specifications for $D_2, \ldots D_n$, but the commitments for the latter may further specify constraints for D_1 as well, a strategy that Steier (1989) has identified as *progressive deepening* is a natural strategy to emerge. This strategy involves making some commitment for each subproblem at each pass, using these commitments to generate additional specifications, undoing earlier commitments as needed, and repeating this process.

Control issues. There are two sets of control issues, one dealing with which sets of decompositions to choose (in Step 1 in the example Decomposition/Recomposition method above), and the other concerned with the order in which the subproblems within a given decomposition ought to be attacked (Step 3). For the first problem, in the general case, the decomposition will produce an AND or an OR node. The decompositions in an AND node will all need to be solved, while for an OR node only one of the decompositions will need to be solved. Finding the appropriate decomposition requires search in an AND/OR graph. But as a rule such searches are expensive. In domains where multiple decompositions are possible but there are no easily formalizable heuristics to choose among them, the machine may be effective in proposing alternatives while the human evaluates them and makes a selection.

In routine design, extensive searches in the spaces of possible decompositions are avoided by limiting the number of possible decompositions at each choice point to one or very few. This leads to the availability of a design hierarchy for design in that domain.

Transformation methods (Balzer, 1981) for algorithm synthesis are examples of decomposition methods. In this approach, a set of high-level specifications of an algorithm is converted into a series of programming-language-level commitments. This is done by mapping subsets of specifications into a "component" for which some implementation-level commitments have been made. Each such commitment will typically constrain other implementation commitments. Because of this, search in the space of possible transformations may often be needed. In most implemented transformation systems, humans choose from a set of alternative transformations presented by the design system.

Regarding the order in which subproblems in a given decomposition are to be attacked, the main constraint is knowledge about dependencies between subproblems that I just discussed. When the subproblems are organized in the form of a design hierarchy, the default control is control *top-down*, but actual control can be complicated. For example, a component at the leaf level of the design hierarchy may be the most limiting component and many other components and subsystems can only

be designed after that is chosen. Part of the design process in this case will appear to have a bottom-up flavour. In general, appropriate control strategies come about based on the dependencies between subproblems.

Design plans. A special case of decomposition knowledge is *design plans*, representing a precompiled partial solution to a design goal (Friedland, 1979; Rich, 1981; Johnson and Soloway, 1985; Mittal *et al.*, 1986; Brown and Chandrasekaran, 1989). A design plan specifies a sequence of design actions to take for producing a design or part of a design. Design commitments made by a design plan may be abstract, i.e. choices are made not at the level of primitive objects but rather intermediate level design abstractions which need to be further refined at the level of primitive objects. For example, in designing an automobile, a design plan may commit to choice of diesel engine as the power plant. While this is a design proposal in the sense that a commitment is being made, the diesel engine design itself is not specified in detail at this stage, but posed as a subtask to be solved by any of the available methods.

Thus a design plan D may set up other design problems $D_1, \ldots, D_n$ as subproblems, and, in this sense, it is decomposition knowledge in a strong form: how the main problem goals are transformed into goals to be allocated to subproblems and how the solutions to the subproblems are put back together for obtaining a solution to the original design problem are all directly encoded in the plan.

Design plans can be indexed in a number of ways. Two possibilities are to index by design goals (*for achieving* ‹*goal*›, *use* ‹plan›), or by components (*for designing* ‹*part*›, *use* ‹*plan*›). Each goal or component may have a small number of alternative plans attached to it, with perhaps some additional knowledge that helps in choosing among them.

Control and inference issues in the use of plans are similar to those in the general case of decomposition: alternative plans are possible and, in routine design, design plan hierarchies may emerge. The default control strategy can be characterized as *instantiate and expand*. That is, the plan's steps specify some of the design parameters, and also specify calls to other design plans. Choosing an abstract plan and making commitments that are specific to the problem at hand is the instantiation process, and calling other plans for specifying details to portions is the expansion part.

A number of additional pieces of information may be needed or generated as this expansion process is undertaken. Information about dependencies between parts of the plan may need to be generated at runtime (e.g. discovering that certain parameters of a piston would need to be chosen before that of the rod), and some optimizations may be discovered at run time (e.g. the same base that was used to attach component A can also be used to attach component B). NOAH

(Sacerdoti, 1975) is an early example of runtime generation of dependencies and optimization.

Design proposal by case retrieval

A major source of design proposal knowledge is design cases—instances of successful past design problem solving. Cases can arise from an individual's problem-solving experience or that of an organization such as a design firm or a design community. Cases can be episodic (i.e. represent one problem-solving episode) or can represent the result of abstraction and generalization over several episodes. Design plans can be considered to be fairly abstracted versions of numerous cases.

Sussman (1973) proposed that a design strategy is to choose an already completed design that satisfies constraints closest to the ones that apply to the current problem, and modify this design for the current constraints. Schank (1982) has emphasized the importance of case-based problem solving in general. Recent work on case-based reasoning in planning and design (Goel and Chandrasekaran, 1989a; Hammond, 1989) explores this family of methods. In case-based reasoning, "almost correct designs" are obtained by searching a memory bank of previous cases for a design that is similar to the one that is currently being sought.

The heart of case-based design proposal is *Matching*: how to choose the design that is "closest" to the current problem? Clearly some features of the cases are more important in matching than others. Some notion of prioritizing over goals or differences, in the sense of *means–ends analysis*, may be needed.

Indexing of cases with a rich vocabulary of features of the case and the goals it satisfies is a key idea in case-based reasoning. Matching and retrieval can be driven by associative processes on these indices. Much of the work in case-based planning has used domain-specific goals to index cases. For the problem of designing engineering artefacts, the design cases need to be indexed in terms of the output behaviours of interest. For example, Goel and Chandrasekaran (1989b) propose that design cases be indexed using their functions. More generally, they show how cases can be indexed by a causal representation that relates the structure of the device to its function, and how this method of indexing can help in retrieval. Goel (1989) has a proposal for how matching and retrieval can benefit from a principled representation for design goals, states for the device, 41 and the substances the device operates with.

Case-based design proposal has a lot in common with the use of analogical reasoning in design. Maher *et al.* (1988) propose that analogical reasoning in design is at the heart of design creativity.

Design proposal by constraint satisfaction

Under fairly strong assumptions, particular classes of design problems can be formulated as optimization, constraint satisfaction or algebraic equation-solving problems. What is common to all these formulations is that the solution lies in a space determined by simultaneous constraints, and specific classes of computational algorithms may be available to locate that space directly. In particular, when the structure of the design is already specified, but parameters are determined by the specifics of a design problem, numerical or symbolic optimization techniques may be useful for design proposal. Linear, integer and dynamic programming techniques have been used to solve design problems formulated in this manner.

Some versions of the constraint satisfaction problem can be solved by constraint propagation. Constraints can be propagated in such a way that the component parameters are chosen to converge incrementally on a set that satisfies all the constraints (Stefik, 1981).

Formally, *all* design can be thought of as constraint satisfaction, and one might be tempted to propose global constraint satisfaction as a universal solution for design. But unless knowledge is employed to reduce the size of the space (such as by decomposing problems into smaller problems), design by constraint propagation can be computationally intractable. Problem decomposition can create subproblems with sufficient small problem-spaces in which constraint satisfaction methods can work without excessive search.

Verification

This subtask involves checking that the design proposal satisfies the functional and other specifications. These are two families of methods for this:

- Attributes of interest can be directly calculated or estimated by means of domain-specific algorithms or formulae (e.g. use of algebraic formula to calculate total weight or cost, or use of finite-element methods to calculate stress distribution). Direct calculation methods are not of much interest from an AI point of view.
- Behaviours of interest can be derived by *simulation*. These behaviours can be checked against requirements.

Simulation takes as input a description of the structure of the system and generates as output the behaviours of interest. The methods used in simulation should mirror the rules by which the behaviour of assemblages of components is composed from the properties of the components.

There are quantitative simulation methods which use equations that directly describe the results of this composition. These equations again are domain-specific. For example, differential equations may be used to describe the behaviour of a reaction in a reaction vessel. The structural description in a proposed design of a reaction vessel can be translated into parameters of the differential equation and the equation simulated to derive behaviours of interest.

There are generic AI techniques for generating behaviour from structure that could be useful for simulation. Qualitative simulation (see Forbus, 1988, for a survey of the current state of the art), consolidation (Bylander, 1988) and functional simulation (Sticklen, 1987) are examples of AI techniques that are available for deriving behaviours given structure. A proposed design can be simulated under various input conditions and the behaviour evaluated. All these techniques take as input a structural description and, using qualitative descriptions of component behaviours and rules of composition, mimic the operation of the device to produce qualitative descriptions of behaviour. Qualitative and quantitative simulation may alternate: a qualitative simulation may identify behaviours likely to be in unacceptable ranges and a more focused quantitative procedure may be used to get more precise values.

Visual simulations. Visual simulation of artefacts is widely used by human designers in verification. Designs are imagined, represented, and communicated pictorially in domains such as architecture and mechanical engineering. (See Goel and Pirolli (1989) for design protocol studies which show the prevalence of images during design.) It is clear that there is a need for pictorial representations and symbolic representations to coexist in design systems. A major use of imaginal representations is in simulation of design proposals, but they play a role as well in making design proposals by analogy with other domains. Little AI research has been done so far on visual representations that have the qualities needed for pictorial reasoning and imagination and that also have the symbolic properties needed for arbitrary referencing and composition by parts. A beginning in this direction is proposed in Chandrasekaran and Narayanan (1990) and use of such representations for simulation is discussed in Narayanan and Chandrasekaran (1991).

Critiquing

Critiquing is the subtask in which causes of failure of a design are analysed: parts of the structure are identified as potentially responsible for the unacceptable behaviour or constraint violation. Critiquing is really a generalized version of the diagnostic problem, i.e. a problem of mapping from undesirable behaviour to parts of the structure responsible

for the behaviour. Modification of design can be directed to these candidates. Of course, localization of responsibility for failure will not always work: the entire approach to the design may need to be changed.

What is needed for criticism is information about how the structure of the device contributes to (or is intended to contribute to) the desired overall behaviour. An AI method that is commonly used for this subtask is dependency analysis (Stallman and Sussman, 1977). This method is applicable if explicit information is available in the form of dependencies, i.e. knowledge that explicitly relates types of constraint or specification violations to prior design commitments. For example, if total weight of a proposed design is higher than the weight limit, domain-specific knowledge is usually available which identifies parts whose weights are both sufficiently large and can be adjusted. Dependencies may be discovered by analysing pre- and post-conditions of design operators. For example, if a certain output behaviour (say, voltage in an electronic device) of a proposed design is excessive, the inputs the output stage can be traced back to identify which of the components upstream may have contributed to the specific output. This analysis may use simulation as a subtask.

Most of the proposals for critiquing that have been in the case-based reasoning literature use domain-specific critics and are variations on pre-compiled patterns of relating output behaviour to possible changes. The approach of Goel (1989) for critiquing a design proposal is based on a functional analysis of the proposed design. If a design proposal is endowed with causal indices that explicitly indicate the relation between structure and intended functions, then it is relatively easy to identify substructures for modification (Goel and Chandrasekaran, 1989a).

Modification

Modification as a subtask takes information about failure of a candidate design as its input and then changes the design so as to get closer to the specifications. Basically, what is required is change of a functional subpart of the proposed design, or addition of components to the proposed design, so as to satisfy the design specifications. Depending on how informative failure analysis is and what types of knowledge are available, a number of problem-solving processes are applicable. Some of them are briefly outlined in the following paragraphs.

Modification may be driven by a form of means-ends reasoning, where the differences are "reduced" in order of most to least significant. Especially useful here is knowledge that relates the desired changes in behaviour to possible structural changes (Goel, 1989).

A related search approach is one where modification is done by some form of hill-climbing. In this method, parameters are changed, direction of

improvement is noted, and additional changes are made in the direction of maximal increment in some measure of overall performance. This is especially applicable where the design problem is viewed as a parameter-choice problem for a predetermined structure (e.g. the Dominic system—Dixon *et al.*, 1984).

Modification is straightforward in dependency-directed methods. Once the dependency point is reached by backtracking, simply an alternative choice is made from the list of finite choices available.

Some systems that perform routine design problems have explicit knowledge about what to do under different kinds of failures. This information can be attached to the design plans (DSPL; Brown and Chandrasekaran, 1989).

Criticism may reveal the need to add new functions. If these functions can be added modularly, i.e. by the creation and integration of separate substructures that deliver the functions, the design of the additional structures can be viewed simply as new design problems to be solved by all the methods available for design. The subtasks of generation of specifications for these additional design problems and integration of their solutions were discussed in the section on problem decomposition and solution recomposition.

DISCUSSION OF THE TASK-STRUCTURE

The task-structure for design described in the preceding section[5] is summarized in Table 1. A task-structure is a description of the task, proposed methods for it, their internal and external subtasks, knowledge required for the methods, and any control strategies for the method. Thus the task analysis provides a clear road map for knowledge acquisition. How the analysis can be used to integrate the methods and goals is discussed in the following section.

Choice of methods. How are methods to be chosen for the various tasks? The following is a set of criteria.

- Properties of the solution. Some methods may produce answers which are precise, while answers of the others may only be qualitative. Some of them may produce optimal solutions, while others may produce satisficing ones.
- Properties of the solution process. Is the computation pragmatically feasible? How much time does it take? Memory?

[5] The task-structure described here is inherently incomplete: additional methods may be identified for any subtask as a result of further research.

Table 1

Task	Methods	Subtasks
Design		
Propose	Propose, Critique, Modify family (PCM)	Propose, Verify Critique, Modify
	Decomposition methods (incl. Design Plans) and Transformation methods	Specification generation for subproblems
		Solution of subproblems generated by decomposition (another set of Design-tasks)
		Composition of subproblem solutions
	Case-based methods	Match and retrieve similar case
	Global constraint-satisfaction methods	
	Numerical optimization methods	
	Numerical or Symbolic constraint propagation methods	
Specification generation for subproblems		
	Constraint propagation, including constraint posting	Simulation to decide how constraints propagate
Composition of subproblem solutions	Configuration methods	Simulation for prediction of behaviour of candidate configurations
Verify	Domain-specific calculations or simulation	
	Qualitative simulation, Consolidation	
	Visual simulation	

Table 1 continued

Task	Methods	Subtasks
Critique	Causal behavioural analysis techniques to assign responsibility	
	Dependency-analysis techniques	
Modify	Hill-climbing-like methods which incrementally improve parameters	
	Dependency-based changes	
	Function-to-structure mapping knowledge	
	Add new functions	Design new function, Recompose with candidate design

For each task, there is a default "compiled knowledge" method which has domain-specific knowledge to achieve it directly and which is not included above. For subtasks such as critiquing, only families of generic AI methods are indicated, without explicit indication of their subtasks

- Availability of knowledge required for applying the method. For example, a method for design verification might require that we have available a description of the behaviour of the device as a system of differential equations; if this information is not available directly and if it cannot be generated by additional problem solving, the method cannot be used.

A delineation of the methods and their properties helps us to move away from abstract arguments about ideal methods for design. Each method in a task-structure can be evaluated for appropriateness in a given situation by asking questions reflecting the above criteria. While some of this evaluation can take place at problem-solving time, much of it can be done at the time of design of the knowledge system; this evaluation can be used to guide a knowledge-system designer in the choice of methods to implement.

Different types of methods may be used for different subtasks. For example, a design system may use a knowledge-based problem-solving method for the subtask of creating a design, but use a quantitative method such as a finite-element method for the subtask of evaluating the design.

IMPLICATIONS FOR AN ARCHITECTURE FOR DESIGN PROBLEM SOLVING

Because of the multiplicity of possible methods and subtasks for a task, a task-specific architecture that is exclusively for design is not likely to be complete: even though design is a generic activity, there is no one generic method for it. Further, note that subtasks such as simulation are not particularly specific to design as a task. Thus if the knowledge for these modules is embedded within a design architecture, either they will be unavailable for other tasks which require simulation as a subtask, or the knowledge for these tasks will need to be replicated. Thus, instead of building monolithic task-specific architectures for such complex tasks, a more useful architectural approach is one that can invoke different methods for different subtasks in a flexible way.

Following the ideas in the work on task-specific architectures, we can support methods by means of special-purpose shells that can help encode knowledge and control problem solving. This is an immediate extension of the generic task methodology (Chandrasekaran, 1986). These methods can then be combined in a domain-specific manner, i.e. methods for subtasks can be selected in advance and included as part of the application system. Alternatively, methods can be chosen at runtime for the tasks recursively, based on the criteria listed above in the paragraph on choice of methods. For the latter, what is needed is a task-independent architecture with the capability of evaluating different methods, choosing one, executing it, setting up subgoals as they arise from the chosen method and repeating the process. Soar (Rosenbloom *et al.*, 1987), BB1 (Hayes-Roth, 1985) or TIPS (Punch, 1989) are good candidates for such an architecture. This approach combines the advantages of task-specific architectures and the flexibility of runtime choice of methods. The DSPL++ work of Herman (1992) is an attempt to do precisely this.

Using method-specific knowledge and strategy representations within a general architecture that helps select methods and set up subgoals is a good first step in adding flexibility to the advantages of the task-specific architecture view. However, it can have limitations as well. For many real-world problems, switching between methods may result in control that is too large-grained. In order to see this, consider my earlier description of a PCM method. The method description calls for a specific sequence of how the operators of Propose, etc., are to be applied. As pointed out in my discussion on the PCM method, numerous variants of the method, with complex sequencing of the various operators, may be appropriate in different domains. It would be a hopeless task to try to support all these variants of the methods by method-specific architectures or shells. It is much better in the long run to let the task–method–subtask

analysis guide us in the identification of the needed task-specific knowledge and let a flexible general architecture determine the actual sequence of operator application by using additional domain-specific knowledge. The subtasks can then be combined flexibly in response to problem-solving needs, achieving a much finer-grained control behaviour. (See Johnson *et al.*, 1989 for realization of generic task ideas in Soar.)

The task structure also makes clear how "AI-like" methods and other algorithmic or numerical methods can be flexibly combined, much as human designers alternate between problem solving in their heads and formal calculations. For example, a designer may need to make sure that the maximum current in a proposed circuit is less than the limits for its components, and at that point he may set up current and voltage equations and solve them. If he finds that the current in one branch of the circuit is more than the permitted limit, he may go back to critiquing the design to look for possible places to change the design. The task-structure view that I have outlined shows how computer-based design systems can also similarly engage in a flexible integration of problem-solving and other forms of algorithmic activity. The key is that the top-level control is goal-oriented, and it can set up subgoals and choose methods that are appropriate to the subgoal. If the appropriate method for a subtask is a numerical algorithm, that method can be invoked and executed, at the end of which control reverts to the top level for pursuing other goals.

CONCLUDING REMARKS

Over the last several years, there have been a number of working systems which perform some version of the design task in some domain. These design proposals do not always bring out what is common among the different tasks of design. There have also been attempts to develop formal "first principles" algorithms for design that are meant to cover all types of design. Such general algorithms are, however, computationally intractable, and are not particularly helpful in identifying the sources of power and tractability in human design problem solving in most domains.

The view elaborated here is that there is a generic vocabulary of tasks and methods that are part of design, and that design problems in different domains simply differ in the mixture of subtasks and methods. Expertise, i.e. methods, and knowledge and control strategies for them, emerge over a period in different domains so as to help solve the task in a given domain tractably. The key to understanding all this is thus not in a uniform algorithm for design, but in the structure of the task, showing how the tasks, methods, subtasks and domain knowledge are related. The analysis also clarifies the relationship between task-specific architectures and more general-purpose architectures for knowledge systems.

ACKNOWLEDGEMENTS

Many ideas from my collaborations with the following have found their way into this chapter: with David C. Brown and Ashok Goel on design problem solving, and with Tom Bylander, John Josephson, Todd Johnson, Jack W. Smith and Jon Sticklen on generic tasks. I thank John Gero, Ashok Goel, Mary Lou Maher, Dale Moberg and David Steier for very useful comments on earlier drafts. The usual caveat holds good that they do not necessarily agree with all of what I am saying in the chapter. Support from AFOSR (grants 87–0090 and 89–0250) and DARPA (contracts F30602–85–C–0010 and F49620–89–C–0110) is gratefully acknowledged.

REFERENCES

Balzer, R. (1981). Transformation implementation: an example. *IEEE Transaction on Software Engineering*, **SE-7**, 3–14.

Brown, D. C. and Chandrasekaran, B. (1989). *Design Problem Solving: Knowledge Structures and Control Strategies*. San Mateo, Calif.: Morgan Kaufmann.

Bylander, T. C. (1988). A critique of qualitative simulation from a consolidation point of view. *IEEE Transactions on Systems, Man and Cybernetics*, **18**(2), 252–268.

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert*, **1**(3), 23–30.

Chandrasekaran, B. (1989). Task structures, knowledge acquisition and learning. *Machine Learning*, **4**, 339–345.

Chandrasekaran, B. and Narayanan, N. H. (1990). Integrating imagery and visual representations. *Proc. Cognitive Science Society Annual Conference, MIT, Cambridge, Mass.*

Chandrasekaran, B., Smith, J. W. Jr and Sticklen, J. (1989). "Deep" models and their relation to diagnosis. *Artificial Intelligence in Medicine*, **1**, 29–40.

Clancey, W. J. (1985). Heuristic classification. *Artificial Intelligence* **27**(3), 289–350.

Dixon, J. R., Simmons, M. K. and Cohen, P. R. (1984). An architecture for application of artificial intelligence to design. *Proceedings of the 21st Design Automation Conference, IEEE*, 634–640.

Forbus, K. D. (1988). Qualitative physics: past, present and future. In *Exploring Artificial Intelligence*. San Mateo, Calif.: Morgan Kauffman, pp. 239–296.

Freeman, P. and Newell, A. (1971). A model for functional reasoning in

design. *Proceedings of the International Joint Conference on Artificial Intelligence*. London: The British Computer Society, pp. 621–633.

Friedland, P. (1979). Knowledge-based experimental design in molecular genetics. *Proceedings of the 6th International Joint Conference in Artificial Intelligence, IJCAI, Tokyo*, 285–287.

Goel, A. (1989). Integration of case-based reasoning and model-based reasoning for adaptive design problem solving. Ph.D. dissertation, Computer & Information Science, The Ohio State University.

Goel, A. and Chandrasekaran, B. (1989a). Functional representation of designs and redesign problem solving. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Mich., 1388–1394.

Goel, A. and Chandrasekaran, B. (1989b). Use of device models in adaptation of design cases. *Proceedings of the Second DARPA Case-Based Reasoning Workshop*, Pensacola, 100–109.

Goel, V. and Pirolli, P. (1989). Motivating the notion of generic design within information-processing theory: the design problem space. *AI Magazine*, **10**(1), 18–38.

Hammond, K. (1989). *Case-Based Planning: Viewing Planning as a Memory Task*. Boston, Mass.: Academic Press.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence*, **26**, 251–321.

Herman, D. J. (1992). DSPL++: A high-level language for building design expert systems with flexible use of multiple methods (tentative title). Ph.D dissertation, Computer & Information Science, The Ohio State University.

Johnson, L. and Soloway, E. (1985). PROUST: knowlcdge-based program under-standing. *IEEE Transactions on Software Engineering*, **11**(3), 267–275.

Johnson, T., Chandrasekaran, B. and Smith, J. W. Jr (1989). Generic tasks and Soar. *Working Notes of the AAAI Spring Symposium on Knowledge System Development Tools and Languages*, Stanford, Calif., 25–28.

Kelly, V. E. and Steinberg, L. I. (1982). The CRITTER system: analyzing digital circuits by propagating behaviors and specification. *Proceedings AAAI Conference, AAAI*, 284–289.

Keuneke, A. (1989). Machine understanding of devices: causal explanations of diagnostic conclusions. Ph.D. dissertation, Computer & Information Science, The Ohio State University.

Maher, M. L., Zhao, F. and Gero, J. S. (1988). Creativity in humans and computers. In J. S. Gero and T. Oksala (eds) *Preprints Knowledge-Based Design in Architecture*. Helsinki University of Technology, pp. 29–44.

Marcus, S., McDermott, J. and Wang, T. (1985). Knowledge acquisition

for constructive systems. *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*, 637–639.

McDermott, J. (1988). A taxonomy of problem-solving methods. In S. Marcus (ed.) *Automating Knowledge Acquisition for Expert Systems*. Boston: Kluwer, pp. 225–256.

Mittal, S., Dym, C. and Morjaria, M. (1986). PRIDE: an expert system for the design of paper handling systems. *IEEE Computer*, **19**(7), 102–114.

Mittal, S. and Frayman, F. (1989). Towards a generic model of configuration tasks. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Mich, pp. 1395–1401.

Narayanan, N. H. and Chandrasekaran, B. (1991). Reasoning visually about spatial interactions. *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia. Mountain View, CA: Morgan Kauffman, pp. 360–365.

Newell, A. (1980). Reasoning, problem solving and decision process: the problem space as a fundamental category. In *Attention and Performance, VIII*. Hillsdale, NJ: Lawrence Erlbaum, pp. 693–718.

Punch, W. (1989). A diagnosis system using a task-integrated problem solver architecture (TIPS), including causal reasoning. Ph.D. dissertation, Computer & Information Science, The Ohio State University.

Rich, C. (1981). A formal representation for plans in the Programmer's Apprentice. *Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI*, Vancouver, B.C., 1044–1052.

Rosenbloom, P. S., Laird, J. E. and Newell, A. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, **33**, 1–64.

Sacerdoti, E. D. (1975). A structure for plans and behavior. *Technical Report 109*, AI Center, SRI, Menlo Park, Calif.

Schank, R. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*. New York: Cambridge University Press.

Stallman, R. and Sussman, G. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, **9**, 135–196.

Steels, L. (1990). Components of expertise. *AI Magazine*, **11**(2), 28–49.

Stefik, M. (1981). Planning with constraints. *Artificial Intelligence*, **16**, 111–140.

Steier, D. (1989). Automating algorithm design within an architecture for general intelligence. Ph.D. dissertation, School of Computer Science, Carnegie Mellon University, CMU–CS–89–128.

Sticklen, J. (1987). MDX2: An integrated medical diagnosis system. Ph.D. dissertation, Computer & Information Science, The Ohio State University.

Sussman, G. J. (1973). A computational model for skill acquisition. Ph.D. dissertation, MIT, AI–TR 197. [Also in book, same title, Elsevier, 1975.]

B. Chandrasekaran, Todd R. Johnson and Jack W. Smith

Task-Structure Analysis for Knowledge Modeling

In recent years there has been increasing interest in describing complicated information processing systems in terms of the knowledge they have, rather than by the details of their implementation. This requires a means of modeling the knowledge in a system. Several different approaches to knowledge modeling have been developed by researchers working in Artificial Intelligence (AI). Most of these approaches share the view that knowledge must be modeled with respect to a goal or task. In this article, we outline our modeling approach in terms of the notion of a *task-structure*, which recursively links a task to alternative methods and to their subtasks. Our emphasis is on the notion of modeling domain knowledge using tasks and methods as mediating concepts. We begin by tracing the development of a number of different knowledge-modeling approaches. These approaches share many features, but their differences make it difficult to compare systems that have been modeled using different approaches. We present these approaches and describe their similarities and differences. We then give a detailed description, based on the task structure, of our knowledge-modeling approach and illustrate it with task structures for diagnosis and design. Finally, we show how the task structure can be used to compare and unify the other approaches.

Knowledge-Based Systems: What are they?

A knowledge-based system (KBS) has explicit representations of knowledge as well as inference processes that operate on these representations to achieve a goal. An inference process consists of a number of inference steps, each step creating additional knowledge. The process of applying inference steps is repeated until the information needed to fulfill the requirements of the problem-solving goal or task is generated. Typically, both domain knowledge and possible inference steps have to be modeled and represented in some form.

In one sense, knowledge is of general utility—the same piece can be utilized in different contexts and problems; so, unlike traditional procedural approaches, knowledge should not be tied to one task or goal. On the other hand, it is difficult to know what knowledge to put in a system without having an idea of the tasks the KBS will confront. In spite of claims of generality, all KBSs are designed with some task or class of tasks in mind. Similarly, they are designed to be operational across some range of domains. Thus, a clear understanding of the relationship between tasks, knowledge and inferences required to perform the task is needed before knowledge in any domain can be modeled.

Background

Tasks

The word "task" has been used in somewhat different senses in the field, contributing to much confusion. For example, Wielinga et al. in [35] describe a task as a "fixed strategy" for achieving a goal, implying that it is a term synonymous with a method or a procedure specification. In our original work on generic tasks (GT) [6], there was a conflation of the goal with the method: the GTs could be thought of as components of a composite method (e.g., the goal of diagnosis is achieved by a method composed of "data abstraction" and "classification"), or they could be thought of as goals as well (the GT called "hypothesis assessment" had the goal to assess hypotheses). In this article, we use the word "task" as synonymous with types of problem-solving goals: for example, we call diagnosis a task, since we want to talk abstractly about the family of problems, all of which are characterized by achieving the goal of generating a causal explanation of observed abnormal behavior. Specifically, we want to separate the task from the method used to achieve goals of this type.

The Knowledge Level

Newell's Knowledge Level (KL) framework [27] is very useful for describing intelligent systems without becoming bogged down in accidental features of implementation languages (see [20] in this issue for additional discussion and examples of the knowledge level). Much of the discussion in the field has been vitiated by too premature a commitment to a symbol-level repre-

sentation (e.g., whether the representation will be rules or frames, and whether backward- or forward-chaining will be employed). Newell proposed that problem-solving agents can be characterized in terms of the knowledge and goals that can be attributed to them, and the Principle of Rationality by means of which intelligent agents can be assumed to use knowledge relevant to their goals. Thus, in discussing a diagnostic system, whether it is implemented as a rule-based system or a connectionist network, we can talk about the task (or goal) as diagnosis of a certain type, and can identify the knowledge content of the system in two ways: as knowledge about the set of malfunctions, and knowledge that aids in mapping from observations to the malfunctions. Hence, at the knowledge-modeling level, we relate the task to the types of knowledge needed to accomplish it. We can then make additional implementation commitments which will, in turn, give us additional constraints on the forms of knowledge.

The knowledge-level view does not include a specific account of how the problem will be solved (i.e., it does not indicate the representations and inference methods utilized to accomplish the task). However, the knowledge-level view can be applied recursively: that is, some commitment to an inference method can be made fairly abstractly: then this too can be represented as knowledge at the knowledge level. This process can be repeated until the knowledge-level description includes some description of the strategies as well. Each commitment to a method requires some commitment to how the problem will be solved, but not as detailed a symbol-level commitment as is normally done when a programming language such as rules or frame languages is employed.

To see how the KL is used, consider how it can be applied to describe MYCIN, a KBS for selecting therapies for bacterial infections of the blood [33]. At the highest knowledge level the goal of MYCIN is to select a therapy. The knowledge required to do this maps signs and symptoms to therapies. At the next level we can reapply the KL and say that MYCIN's therapy goal is accomplished by first identifying the bacterial infection present, then selecting a therapy for that infection. Hence, we break the top goal, therapy, into two subgoals, diagnosis and selection. At this level we can be more specific about the types of knowledge required for the task. Diagnosis requires knowledge-mapping signs and symptoms to an infection. Selection requires knowledge mapping infections and patient data to a therapy. In such a way, we can continue to apply the KL until the system is specified in sufficient detail to allow its implementation. In MYCIN, each of the subtasks is implemented in a backward-chaining rule-based system. The point, however, is that an accurate KL description of MYCIN hides implementation details of the system.

There are many ways to specify an information processing system without describing implementation details. Examples include the knowledge level, abstract algorithm specifications and Marr's information processing level [22]. The selection of an information processing description depends largely on the system being described and the purpose of the description. The KL is primarily designed for use in describing intelligent agents; hence it describes an information processing system as having goals, actions and bodies of knowledge.

Background Work in Knowledge Modeling

Some of the earliest work in knowledge modeling was done as part of the rule-based system approach. In this approach, the agent's knowledge was viewed mainly as directly available recognition knowledge (i.e., knowledge that indicates exactly what to do in a situation). The knowledge modeling scheme was simply to list, as knowledge, the condition-action rules by which a system behaves. Simple condition-action statements, in which the conditions match the current situation and the actions add to or modify that situation, are termed production rules. However, this level of description does not indicate the real control structure of the system at the task level. For example, the fact that R1 [23] performs a linear sequence of subtasks is not explicitly encoded; the system designer "encrypted," so to speak, this control in the pattern-matching of OPS5, the production-rule system in which R1 is implemented.

Another early knowledge-modeling scheme was based on frames. Frames were often proposed to be at the "knowledge level," since they supposedly were used for representing objects in the domain and their relations, a "deeper" level of representation than production rules. The knowledge-level idea behind frames is that they capture stereotypical knowledge; this idea, however, is not sufficient for modeling control knowledge at the task level. The problem is that frames and frame languages do not provide a task-level vocabulary for modeling control knowledge. When frame languages were used, control of a system was often described at a syntactic level: for example, in terms of which links to pursue for inheritance.

The problem was that during the first decade of knowledge-based systems research, the discussion was almost entirely in terms of the symbol level: in the rule-based paradigm all the problems were posed as issues at the rule architecture level. Very little discussion took place at the level of the relation between the task for which the system was being designed and the kinds of knowledge needed. For example, a major research issue for rule-based systems was the development of an appropriate domain- and task-independent *conflict resolution strategy* that would let the system choose which production rule to

fire when multiple rules matched. When the knowledge is viewed at the appropriate level, however, we can often see the existence of organizations of knowledge that bring up only a small, highly relevant body of knowledge without any need for conflict resolution.

The first set of insights regarding the analysis of knowledge systems at one level removed from their implementations came from a number of sources. Gomez and Chandrasekaran identified classification as a common element in diagnosis [15], and Mittal and Chandrasekaran added data abstraction as another common element [25]. This work led directly to a knowledge-modeling scheme in the form of generic task (GT) languages [6]. A generic task identifies a task of general utility (such as classification), a method for doing the task and the kinds of knowledge needed by the method. The language is made of primitives that allow the required knowledge to be directly described for any domain in which the task can be performed. Chandrasekaran and his colleagues identified a number of such generic tasks [6]. They also showed by example how complex tasks such as diagnosis could be decomposed into such generic tasks [9].

Hierarchical classification [5, 9] was the first generic task to be identified and will serve as a good example of the task-based approach that we and other researchers have been developing. The task of hierarchical classification is the identification of an object based on a set of features. For example, diagnosis can be viewed as classification in which the input is a set of manifestations and the output is the disorder associated with these manifestations. The name of the generic task, hierarchical classification, alludes to the method identified to solve the task. The method, called *Establish-Refine*, assumes the existence of a classification hierarchy of output categories. In the case of medical diagnosis, the hierarchy contains diseases. More general classes of diseases are located at the top of the hierarchy; more specific diseases are located near the bottom. The method operates by first attempting to establish (i.e., confirm to a certain level of confidence) the topmost category. If this can be established, it is then refined: its successors become category hypotheses for the system to consider next. Categories that cannot be established are not refined; hence the hierarchy below these categories does not need to be explored. The generic task description clarifies the control structure and knowledge of a classification system. Instead of describing the system in terms of rules or frames, we can describe the system in terms of categories, category evaluation and refinement. The knowledge required to use hierarchical classification is made explicit: knowledge must be available to test and refine categories. The control of the system is explicitly described at a task level—categories are evaluated and then (if necessary) refined—rather than at the implementation or symbol level.

Somewhat near this time, Clancey had identified "heuristic classification" [11] as a somewhat abstract pattern of inference implicit in MYCIN (see Figure 1). Heuristic classification itself was presented as independent of the rule language in which MYCIN was written so that this higher-level inference pattern could be seen independent of the rule-level representation. Clancey's approach is similar to the GT approach, having identified a task (classification), a method (heuristic classification) and the kinds of knowledge needed to use the method. In fact, the three inferences in heuristic classification (see Figure 1) can be interpreted as three subtasks. MDX [9], the generic-task diagnosis system, also incorporated the same task decomposition. Data abstraction was done using an intelligent database; heuristic match was done by establishing categories in the classification hierarchy, and refinement was done during classification. The difference between the two approaches lies in the explicit identification by Clancey of this combined inference structure as heuristic classification, whereas Chandrasekaran et al. had broken this structure down into its components.

McDermott and associates started investigating the roles of knowledge in various methods for several tasks [24]. Their goal was to develop programs that could automatically acquire knowledge from a domain expert. To do this they developed *role-limiting methods* for solving general tasks, such as the cover-and-differentiate method for diagnosis and the propose-and-revise method for design. Role-limiting methods are "methods that strongly guide knowledge collection and encoding [24]." They proceeded to specify the roles various types of knowledge play in the operation of each method. The major difference between the role-limiting method approach and most of the other approaches discussed here is the requirement that a role-limiting method be completely specified (i.e., that all tasks and subtasks be prespecified down to the level of primitive operations).

Musen investigated ways to model classes of planning problems and the required domain knowledge [26]. He advocated the development of a task model followed by the use of the model to acquire domain facts. His system, Protegé, provides a language for modeling classes of planning problems based on skeletal plan refinement. Once modeled, Protegé created a knowledge editor domain which experts could interact with to build KBSs that solved problems in the planning class.

Gruber and Cohen investigated task models as mediating representations of knowledge acquisition for a diagnostic task [16]. They constructed MU, a task-specific architecture for building application programs that do prospective diagnosis. A companion system, called

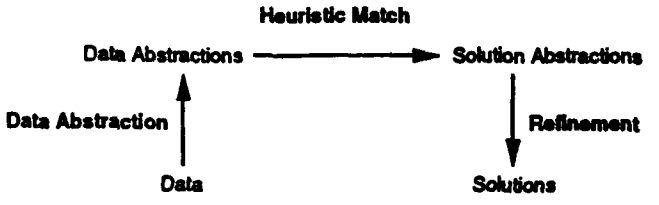


Figure 1. Inference structure for heuristic classification. Adapted from [11].

ASK for "Acquisition of Strategic Knowledge," interacts with an expert to acquire knowledge for MU-based systems. Because ASK is written specifically for MU, it knows about the strategy and types of knowledge needed for the task; hence ASK can interact with an expert at the task level rather than at a lower implementational level.

In Europe, Wielinga and Breuker proposed a set of primitive terms in which to model the tasks that expert systems perform and, in turn, the use of these terms as a modeling language to capture the knowledge in the domain [1]. Their methodology, called KADS, advocates a bottom-up approach to expert knowledge modeling where the knowledge modeler begins with an expert verbal protocol, models this with primitive terms, then builds higher levels of analysis on top of the primitive model. This is quite different from the methodology behind GTs in which the knowledge modeling takes place in a top-down fashion by matching known generic tasks to the task being modeled.

More recently, Steels has proposed work along lines that build on the notion of tasks and task structures [34]. In his formulation, the task structure is intended to specify the task/subtask decomposition of a complex task such as diagnosis. There is a clear recognition that the subtasks of a task depend on the method used for the task. For example, a task such as diagnosis might be done using a classification method. Classification specifies additional subtasks, such as evaluating and refining hypotheses. The task structure notion of Steels does not explicitly represent alternate methods for each of the tasks; instead it is a tree of tasks and subtasks, with the method chosen implicit in the analysis. Thus a given task structure implicitly assumes the choice of some method for a given task. Therefore, it is a good tool for the description of how a particular knowledge system solves the task for which it is intended. The notion of the task structure we will develop later in this article explicitly represents the methods for each task, which then provides a framework for the dynamic selection of methods at run time.

These approaches share two important features. First, they identify tasks at various levels of abstraction above the implementation language level. Second, they identify types of knowledge and strategies closely associated with such tasks. This is the key point in knowledge modeling: once such terms are identified, we have a language in which to model the knowledge in the domain and the strategies to solve the problem. The terms in the vocabulary can be used to encode knowledge, mediate knowledge acquisition [4] and provide suitable explanations [10].

Need for Uniform Framework

In spite of the last decade having seen a clear consensus in favor of task-level analyses and the advantages they offer for knowledge modeling and acquisition, confusion remains regarding the following:

1. Distinctions between tasks and methods and how complex generic tasks and simple generic tasks are related.
2. The great variety of knowledge-modeling terms that have been proposed without any simple way to map between them,
3. Avoiding overdetermination and rigidity in the ways various tasks are performed in the various proposals. That is, we need to show how these various task-level ideas, at various grain sizes, can be combined flexibly.

To overcome these problems, we develop the notions of a task, method, subtask, and the concept of a task structure. The task structure is a uniform task-level analysis framework for describing systems. By viewing the various task-level approaches in terms of the task structure we can begin to compare the approaches and also unravel the current confusions.

The Task Structure

The Task Structure is the tree of tasks, methods and subtasks applied recursively until tasks are reached that are in some sense performed directly using available knowledge. Figure 2 graphically represents part of the task structure for diagnosis. A task (as we defined earlier) is a problem type, such as diagnosis. Tasks are represented graphically using circles. A method is a way of accomplishing a task. These are represented graphically using rectangles. In the figure, *Bayesian Explanation*, *Abductive Assembly* and *Cover-and-differentiate* are identified as methods for diagnosing. All of these methods can be classified as abducuive methods,[1] hence they appear as a subtype of *Abduction Methods*. In general, a task can be accomplished using any one of several alternative methods; thus in the task structure we can explicitly identify alternative methods for each task. A method can set up subtasks, which themselves can be ac-

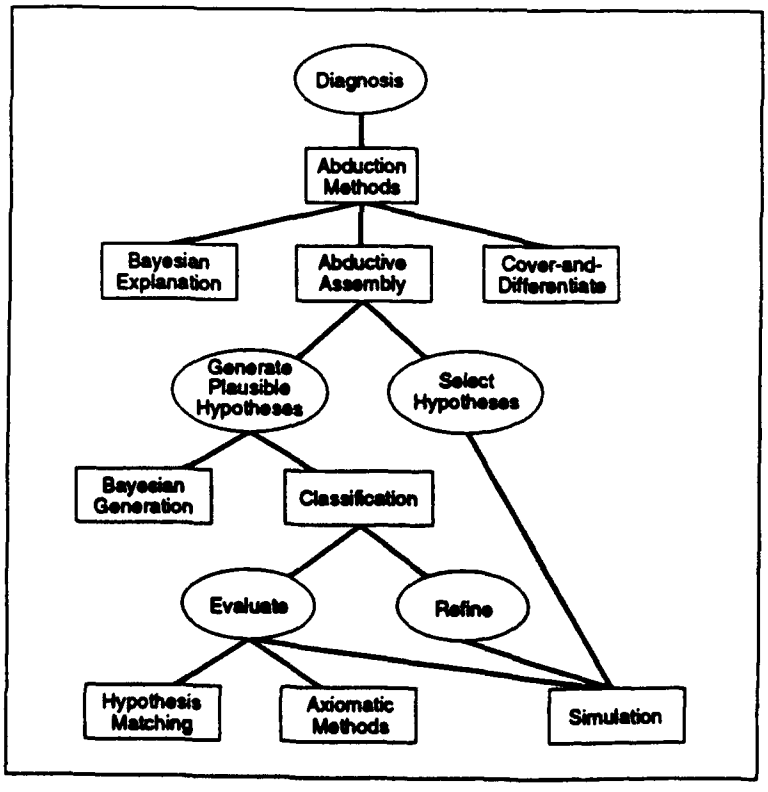
[1] Abduction is the problem of reasoning from effect to cause.

complished by various methods. For example in the *Diagnosis* task structure *Abductive Assembly* has been decomposed into two subtasks: *Generate Plausible Hypotheses* and *Select Hypotheses*.

Knowledge in a task structure comes in four forms. First, each task must be accomplished using knowledge that maps the input of the task to the output. Second, knowledge must indicate when an applicable subtask is needed. Third, when a method consists of subtasks, knowledge is needed to sequence the subtasks. Fourth, when a task can be accomplished using two or more methods, knowledge is needed to select a method.

The following subsections describe the task structure in detail. The section "Examples of Task Structure for Design and Diagnosis" discusses the task structures for design and diagnosis in detail.

Figure 2 Part of the task structure for Diagnosis. Circles represent tasks; rectangles represent methods. See section "Examples of Task Structure for Design and Diagnosis" for a discussion of the role of simulation.



Tasks

Tasks are specified as transforming an initial problem state with certain features to a goal state with certain additional features. For example, in the diagnostic task the initial state includes malfunction observations and the goal state includes information about the causes of the malfunctions. It is important to distinguish between a task and a task instance. A task instance is a particular problem/goal state pair, such as the diagnosis of a particular patient with specific symptoms. In contrast, a task specifies a family of task instances of a certain type. This family can be defined at various levels of generality. For example, the diagnosis of a patient with specific symptoms is a task instance of the medical diagnosis task, which is itself a subclass of the general diagnosis task.

Methods and Subtasks

Methods are ways of accomplishing tasks and may be of many types: they may be computational, or "situated," (i.e., involve extracting information from the surrounding physical world). For example, the task of predicting behavior of a device may be solved by a computational method that performs a simulation, or it may be solved by manipulating a physical model of the device and seeing what happens. Within the class of computational methods, a method may be couched as executing a precompiled algorithm, search in a state space, as a connectionist network and so on.

Our uniform framework for describing methods is based on the problem-space computational model[2] [28] and was adopted as a result of work on TIPS [31], an architecture for dynamically integrating generic tasks, and work done integrating generic tasks using problem spaces in the Soar architecture [18]. We define a method to be a set of subtasks that can be used to transform the initial state of a task to the goal state. The method may contain additional information about ordering the subtasks, called *search-control knowledge* [21], or this knowledge can be generated at run time.

While the task structure allows the specification of methods of different types, those that are modeled as problem-space search have a special role for two reasons. For one thing, one way to understand knowledge systems as a distinct type of information technology is to note that the role of explicit knowledge in them is to set up alternatives, evaluate and refine them. In MYCIN, for example, the knowledge in its knowledge base enables it to set up and evaluate various bacterial infection hypotheses. Second, the architecture that integrates the different methods itself can be viewed as operating in a

[2]For an example of a direct application of the problem-space computational model see [20] in this issue.

search space of methods and making selections in it. Third, we will see that the notion of subtasks emerges naturally in the framework of search in problem spaces.

To clarify this, let us consider how to represent the *Establish-Refine* method for hierarchical classification (see earlier subsection "Background Work in Knowledge Modeling") using the framework described. Hierarchical classification is used in many diagnosis systems as a way of quickly focusing on possible malfunctions. The initial state of the classification task is a set of data (e.g., manifestations in a diagnosis task) and an initial high-level hypothesis (e.g., liver disease). The goal state is one containing plausible malfunction hypotheses (i.e., the most detailed hypotheses consistent with the data). The method works by first considering a high-level malfunction category, such as liver disease, to determine if the malfunction appears likely given the data at hand. If it appears likely, then the malfunction is refined to more specific diseases, hepatitis and cancer, for example. The more specific malfunctions are then evaluated against the data and any that appear likely are refined. This process continues until no more malfunctions can be refined.

We can specify this method using two subtasks:

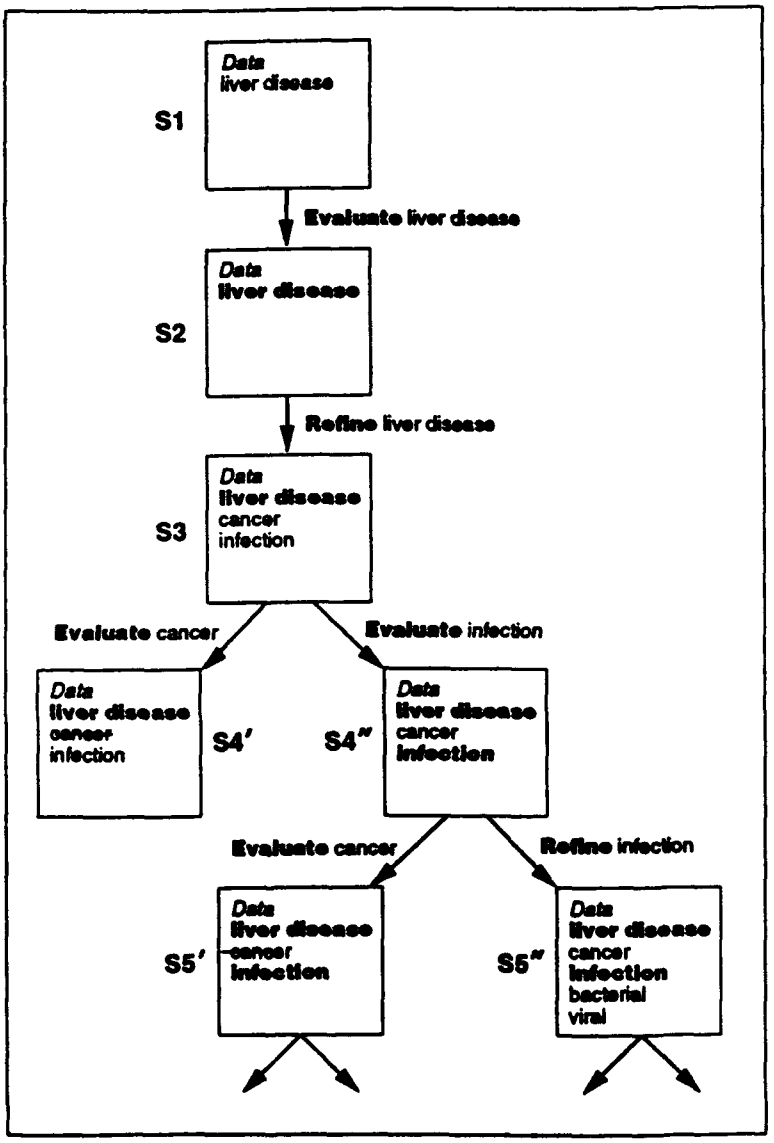
evaluate *hypothesis*
refine *hypothesis*

The first, *evaluate*, takes some hypothesis (such as a malfunction hypothesis) and assigns a likelihood based on the current case data. A precondition for applying *evaluate* to a hypothesis is that the hypothesis must not have already been evaluated. The second subtask, *refine*, takes a hypothesis as input and produces the refinements for that hypothesis. *Refine* has two preconditions: the hypothesis must be likely and must not have already been refined.

We must also specify when an operation should be considered for application to a state. For the hierarchical classification method we are describing, the operations *evaluate* and *refine* should be considered whenever their preconditions are met.

The initial and goal states and the subtasks define a search space or problem space. Figure 3 illustrates the search space that results when the method described is applied to a liver diagnosis problem. The search space is the set of states reachable from the initial state by applying the operators for the method. The figure shows part of the search space of the task, beginning with the initial state, S1, containing manifestations indicative of a viral infection (labeled *Data*) and the high-level hypothesis *liver disease*. The only operator applicable to this state is *evaluate liver disease*. Application of this operation re-

Figure 3. Part of the search space for the hierarchical classification method as applied to liver disease. The data for this example are indicative of a viral infection. Hypotheses in plain text are unevaluated; those in bold are likely; those shown in strike-through are unlikely.



sults in a new state, S2, in which liver disease is rated likely (in the figure this is noted by setting the hypothesis in **bold face**). Only one operator is applicable to S2, *refine liver disease*, resulting in S3 which contains the refinements of *liver disease: cancer* and *infection*. At S3 two operators are applicable: *evaluate cancer* and *evaluate infection;* hence the tree branches to show both possibilities: *evaluate cancer* results in S4' in which *cancer* is determined to be unlikely and *evaluate infection* results in S4" in which *infection* is rated likely.

In the PSCM framework, problems are solved by searching through a problem space for a path from the initial state to the goal state. Problem-space search is done by enumerating subtasks applicable to the current state (which at the start of problem-solving is the initial state of the task instance), selecting from these a single subtask and then applying that operation to the current state. The resulting state then becomes the new current state and the whole process of operation selection and application is repeated until the goal state is reached.

Search-control knowledge guides the search through the problem space. For example, in hierarchical classification the agent might apply a heuristic that it is better to evaluate hypotheses with higher likelihoods than those with low likelihoods, or it might decide that the decision about which evaluate operator to apply is not important, hence either operator can be selected. In the task structure, we specify the minimum amount of search knowledge needed for each method. No search control knowledge is specified for the hierarchical classification method because any such knowledge would unduly constrain the method. For example, if either of the heuristics mentioned previously were included in the search-control knowledge for the method, it would limit the application of the method to those domains and task instances in which the heuristics apply.

The independent specification of search control knowledge and subtasks lead to two of the primary advantages of the problem-space approach to specifying methods:

1. In the task structure we are not forced to specify a particular subtask sequence. We can specify the search-control knowledge that is general to all task instances for a method and defer other decisions about subtask sequencing to system designers or run-time computation. By doing this, we ensure that the method can be applied to as wide a range of task instances as possible. In contrast, early GT work often overconstrained the sequencing of subtasks, limiting the use of each method to a narrow range of problems.
2. Search control knowledge ensures a dynamic or situated system. Each bit of search control knowledge is sensitive to the subtasks and the current state, hence the precise sequence of subtasks is determined dynamically at run time.

Method Selection Knowledge

Functionally, there are four types of knowledge in a task structure. We discussed three of these (search control, subtask application and subtask proposal knowledge) in the previous subsection, since they are related to the description of a method. The remaining type, method selection knowledge, is associated with a task, or a task/method combination. For example, the task structure for Diagnosis includes multiple methods for evaluating a hypothesis. When a system has two or more of these methods, method selection knowledge must be present to determine the best method to take for the task instance being solved.

Direct vs. Derived Knowledge

The four types of knowledge in a task structure can be available in two forms: it can be directly available for the task or it can be computed by another method. Directly available knowledge is in a form that maps the input of the task to the output. For example, directly available knowledge for *refine* is of the form:

> If task is *refine hypothesis* then refinements are *r1*, *r2*, *r3* . . .
> For instance:
> If task is *refine liver disease* then refinements are *infection* and *cancer*.

No complex computation is required to use this knowledge to accomplish the refine task—the knowledge is in a form directly applicable to the task. If knowledge is not directly available, it must be derived from existing knowledge or acquired from the external environment. In either case, a method must be used to acquire knowledge of the desired form. For example, *refine* can be accomplished using a method that knows about different refinement dimensions, such as refinements along etiologic and subpart relations. This method can evaluate various dimensions and then select the dimension appropriate for the task instance. For example, liver disease could be refined using etiology to *infection* and *cancer* or using anatomic structure to *central-area* and *portal area*. The most appropriate dimension to use depends on the task instance (i.e., the kinds of manifestations available).

Whenever any of the four types of knowledge is not directly available, subtasks to acquire the knowledge can be created and set up as new problems. These subtasks are viewed like any other task: they have an initial state and a goal state and can be accomplished by the application of a method consisting of a set of operations. Hence, although a method requires certain kinds of knowledge to be applied to a task, this knowledge does not have to be known before problem-solving can begin, but can be dynamically acquired or derived at run-time.

This idea is also closely related to the distinction between "deep" and "shallow" knowledge, sometimes

called "deep" and "compiled" knowledge. There is also often another distinction between model-based and rule-based reasoning, models being more general knowledge describing the principles of the domain, while rules refer to relatively *ad hoc* associations between evidence and hypotheses. In [8], we provide an analysis of these terms and develop a notion of "depth" of knowledge that is important for knowledge modeling. We give a brief description of this idea.

Let $K(T,M)$ denote the knowledge needed by method M in performing the task T. If a knowledge system performing T using M has the knowledge $K(T,M)$ directly available in its knowledge base, let us say the knowledge system has the knowledge in a *compiled* form. However, suppose some knowledge element k in $K(T,M)$ is missing in the knowledge base, and the task of generating this knowledge is set up as a subtask. If there exists some other body of knowledge in the knowledge base, say K', so that by additional problem-solving using K' we can generate the knowledge element k, we can say that K' is *deep* relative to k.

In the *refine* example we saw that *anatomic structure* is one of the dimensions along which refinement could be done. So-called model-based reasoning is an approach in which structural descriptions of the device under diagnosis are used to generate refinement hypotheses. From this device model, we can generate a list of malfunctions (e.g., one malfunction category can be assigned to the failure of each of the functions of each component; moreover, malfunction categories can correspond to errors in connections between components). The same structural model can be used to generate knowledge needed for the evaluation subtask in Figure 2. The structural model can be simulated for each malfunction, and information about the relation between malfunctions and observations, which is the type of knowledge needed for the methods of the evaluation subtask, can be generated (see the next section for information on simulation). Thus the structural model is a deep model for the methods of classification and hypothesis evaluation that are generally used in the diagnostic task.

The approach to defining the notion of depth of knowledge in the framework of the task/methods/knowledge triple generalizes the intuitive notion that has equated structural models with deep models. Under our definition depth is a relative notion (i.e., it is relative to a method for a task), and there is no notion of characterizing knowledge as deep or shallow in some absolute way.

Examples of Task Structure for Design and Diagnosis

The specification of a task structure consists of three parts:

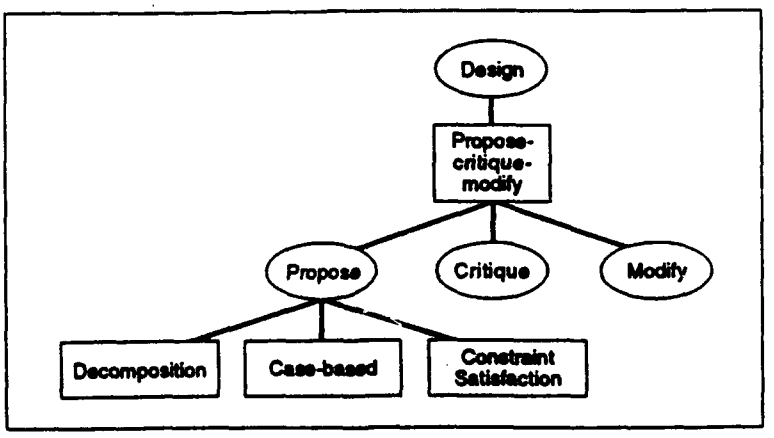
1. an input-output relation that denotes the task;
2. the identification of methods and their subtasks (as in Figure 2); and
3. knowledge to propose subtasks, implement subtasks, sequence subtasks (search-control knowledge) and select methods.

The task-structure diagrams do not list the kinds of knowledge or the input-output relations of the tasks; this is, however, an important part of the specification of the task structure. The following descriptions of design and diagnosis illustrate the main points about specifying the task structure.

Part of the task structure for design is shown in Figure 4. In the task structure diagrams, circles represent tasks and rectangles represent methods. The top task for the design task structure is, of course, design. The design task can be solved using a family of methods called *propose-critique-modify* (PCM) [7]. These methods have the subtasks of proposing partial or complete design solutions, critiquing the proposals by identifying causes of failure, if any, and modifying proposals to satisfy design goals; hence the three subtasks shown for PCM: propose, critique and modify. These subtasks can be combined in fairly complex ways, but the following method is one straightforward way in which a PCM method can organize and combine the subtasks.

Step 1. Given design goal, propose solution. If no proposal, exit with failure.
Step 2. Verify proposal. If verified, exit with success.
Step 3. If unsuccessful, critique proposal to identify sources of failure. If no useful criticism available, exit with failure.
Step 4. Modify proposal; return to 2.

Figure 4 Part of the task structure for design [7]. Circles represent tasks; rectangles represent methods.



There can be numerous variants on the way the methods in this class work. For example, a solution can be proposed for only a part of the design problem, a part deemed to be crucial. This solution can then be critiqued and modified. This partial solution can generate additional constraints, leading to further design commitments. Thus, subtasks can be scheduled in a fairly complex way, with subgoals from different methods alternating. One could generate all such variations and identify them all as distinct methods, but both the need for descriptive parsimony and the sheer numerousness of the methods that would result argue against doing that.

Each of the PCM subtasks can be achieved using various methods. Three such families of methods are shown for the proposal task (see Figure 4): decomposition, case-based and constraint satisfaction. In decomposition methods, domain knowledge is used to map subsets of design specifications into a set of smaller design problems. The use of design plans is a special case of the decomposition method. Case-based methods are those retrieving from memory cases with solutions to design problems similar or close to the current problem. Constraint-satisfaction methods use a variety of quantitative and qualitative optimization techniques.

Part of the task structure for diagnosis is shown in Figure 2. The diagnosis task can be viewed as an abductive task, the construction of a best explanation (one or more disorders) to explain a set of data (manifestations). The task structure shows three typical subclasses of abductive methods: Bayesian, abductive assembly [19] and parsimonious covering [30]. Bayesian methods require knowledge of prior probabilities of disorders and conditional probabilities between disorders and manifestations. They use this knowledge to estimate posterior probabilities of disorders. Abductive assembly requires knowledge of disorders and the manifestations they explain. This method works by first generating plausible hypotheses to explain parts of the data and then using these hypotheses to assemble a complete explanation of the data. Parsimonious covering works by stepping through each manifestation, updating the current set of parsimonious explanations as each manifestation is considered. Two subtasks for abductive assembly are shown in the diagram, *generate-plausible-hypotheses* and *select-hypotheses*. These tasks can be done using many kinds of methods. Since Bayesian and classification methods have typically been used to generate plausible hypotheses, these are shown in the task structure.

The task structure for diagnosis also shows that simulation can be used to implement many subtasks. By simulation we mean structure-to-behavior simulation, that is, determining how some device will behave under changes to its structure by simulating its behavior under those conditions. We have earlier discussed the role of simulation for accomplishing subtasks (see previous subsection "Direct vs. Derived Knowledge").

The knowledge required to use abductive assembly consists of control knowledge for sequencing subtasks as well as the knowledge required to accomplish the subtasks. Control knowledge is specific to an application or domain, but the knowledge for accomplishing subtasks can be defined using the input/output specifications of the subtasks. *Generate-plausible-hypotheses* takes as input one or more manifestations and outputs one or more disorders that could be used to explain those manifestations. *Select-hypothesis* takes as input the set of manifestations, the set of disorders currently being used to explain manifestations, and the set of disorders that could be used to explain one or more additional manifestations (i.e., the output of *generate-plausible-hypotheses*). The output of *select-hypothesis* is the disorder it has determined to use to explain the manifestations. Hence, by describing the input/output of the subtasks of abductive assembly we also specify the knowledge required to use the method. Simulation can be used to evaluate a hypothesis because the simulation can reveal whether the hypothesis is possible given the data about the device. Causal refinements of a category can be determined by simulating to determine the possible outcomes of a set of inputs to a device. Simulation plays an important role in many task structures because it is a fairly general method for generating knowledge based on the structure of a device. We did not show the simulation method in the design task structure, but there too it can play an important role, especially for critiquing designs.

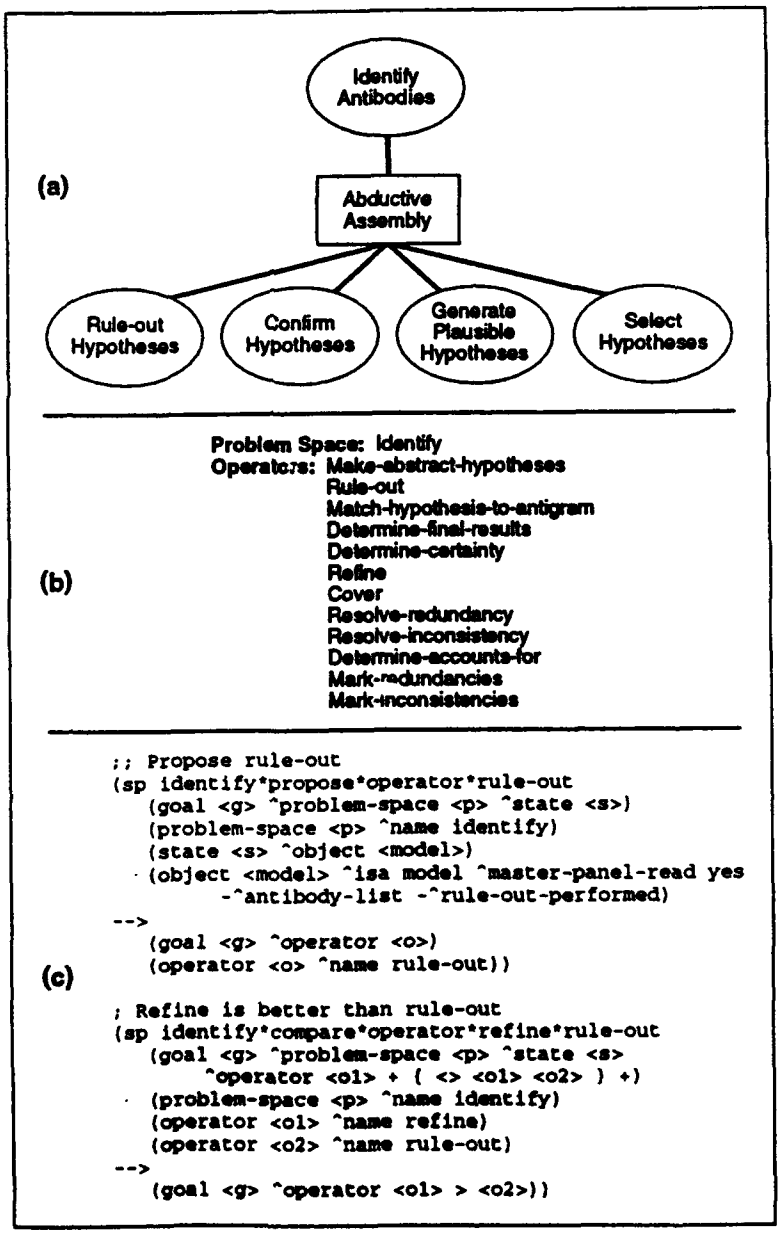
These task structures are based on the methods and subtasks implicit in many expert systems that perform the tasks. Neither of the task structures is meant to be complete; both, however, capture a wide range of the methods useful for achieving the respective tasks. As we discover additional methods, these can be added to the structure. Some methods (such as depth-first search) are so general they can be used to solve any problem. These methods are not listed in the task structure since they would appear everywhere, cluttering the diagram.

The task structure is meant to be an analytical tool. We do not mean to imply that the implementation of a system must have a one-to-one correspondence to the task structure, but that a system that performs diagnosis or design can be *viewed* as using some of the methods and subtasks. In particular, the task structure does not fix the order of subtasks or dictate that a single method must be used to achieve each task. It is also not meant to correspond to a procedure-call hierarchy, although that is one way to directly implement a task structure. The task structure simply provides a vocabulary to use in describing how systems work. The systems

being described might be based on neural networks, production rules, frames, or a task-specific language; this is unimportant for the use and construction of the task structure.

To further emphasize the importance and role of the task structure for describing systems, let us consider three descriptions of RedSoar, a complex abductive system that interprets immunohematologic tests in order to identify antibodies present in a patient's blood [17]. In describing a complex knowledge system such as RedSoar, we can use three levels: 1) the task structure; 2) a computational level (such as problem spaces); and 3) a symbol level. Figure 5 shows each of these levels for RedSoar. RedSoar uses abductive assembly of antibody hypotheses to construct a best explanation of the test data. In the task, the test data are the manifestations; antibodies are the "disorders" or explanations. RedSoar is directly implemented in Soar's production-rule language and can be described by listing all the rules in the knowledge base, such as those in Figure 5c. About 1,000 of these rules constitute the symbol-level view of RedSoar. However, this description fails to capture the task-level control and knowledge in the system. To do this, RedSoar can be described at a computational level by listing the problem spaces defined by the Soar production rules, as in Figure 5b. That is, we can abstract away from the symbol-level production rules to focus on the problem spaces, their initial and desired states and their operators. This level of description is much closer to the task level, but would still contain too many details present as artifacts of the implementation (e.g., extra operators that must be used for low-level manipulation of representations). At the task-structure level (see Figure 5a), we can simply describe the system as using abductive assembly and then point out how it generates and selects hypotheses: i.e., the methods and knowledge that it uses. RedSoar uses conditional and *a priori* probabilities to generate plausible hypotheses and a scoring function based on explanatory coverage and plausibility ratings to select a hypothesis. As shown in the figure, RedSoar also uses two additional subtasks, *rule-out* and *confirm hypotheses*. These are domain-specific subtasks. The first allows the system to quickly rule out clearly absent antibodies. The second lets the system focus on antibodies that are likely present. By describing RedSoar at this level, a comparison can be made between it and other abductive assembly systems by comparing the methods and knowledge used to generate and select hypotheses.

Figure 5 RedSoar described at three levels: a) the task structure; b) the problem-space level (a computational level); and c) production rules (the symbol level).



Knowledge Modeling and the Task Structure

The task structure described in the previous section facilitates knowledge modeling in several ways. First, it associates tasks with methods that accomplish them and the knowledge required to use the methods. The multiple levels of the task structure show how knowledge can be decomposed into bodies of knowledge that are associated with specific tasks. The task structure also highlights the generality and specificity of the knowledge needed for a problem-solving method. That is, it allows methods to be compared based on the required knowledge. Hence, we can see how some methods require little domain knowledge (such as depth-first search, which only requires knowledge to recognize a goal state), while others require considerable domain knowledge (such as hierarchical classification, which needs a domain-specific hierarchy of categories).

Second, since methods are characterized by the knowledge they require, domains can be modeled by tools appropriate for the knowledge that is available in the domain. High-level tools based on this concept, such as CSRL [5], DSPL [2], MUM [16], and MOLE [13], illustrate how this approach facilitates knowledge modeling, knowledge acquisition, explanation, and learning.

Third, the task structure view should be contrasted with what one might call a "uniform normative algorithm" view of how to solve complex problems such as diagnosis or design. For example, there have been proposals for a general algorithm for diagnosis: "diagnosis from first principles" [32], and Bayesian networks [29] are two examples. The general algorithms, while guaranteeing an optimal solution within their respective frameworks, are typically intractable. In these cases the engineering of systems to solve the tasks is done by various forms of heuristic approximations, which of course no longer have the normative properties associated with the original algorithm. The general algorithms also do not always make contact with the form in which knowledge is actually available in various real-world domains. Thus, the Bayesian framework may be fine for a domain in which the needed prior and conditional probabilities (or good approximations to them) are available, but in other domains in which the domain knowledge takes other forms, there is often a need for translating from these forms to the probabilistic forms in which knowledge is needed.

The task structure view, on the other hand, views the solution of complex problems as arising from the interaction of many local methods for local tasks. In any domain having a record of successful human problem-solving, the knowledge in the domain helps to decompose the task into manageable chunks, so each of the problems can be solved to the degree of precision and accuracy needed for the domain. It then becomes the task of the AI theorist to develop vocabularies of generic tasks, methods and knowledge. Thus the attention is shifted from the search for uniform algorithms to modeling knowledge and methods by which tasks are decomposed and subtasks are accomplished.

We can also see how such task structures evolve in real-world domains. If classification is a generally effective method for the generate-hypotheses subtask of diagnosis, then over time, the problem-solving community develops the knowledge needed to apply it. Thus the medical community has devoted hundreds of years to the development of disease taxonomies, which is the form in which the classification method needs knowledge. The knowledge compilation techniques (see subsection "Deep vs. Derived Knowledge") are also a means by which knowledge in a less direct form is converted into knowledge in a form that is more directly usable by a computationally attractive method. Thus we see that in domains and tasks of importance, the domain knowledge tends to *evolve* over time so that methods with good computational properties can be supported.

The fact that we do not start with a uniform normative algorithm does not mean we cannot be precise about the behavior of systems built in the task-structure framework. Bylander [3] and Goel [14] are examples of analyses in which the role of specific types of knowledge in producing good computational properties can be studied within the general framework of the task-structure view. For example, Goel et al. show why classification is an attractive method, if knowledge in the form of classification hierarchies is available, and Bylander et al. show how knowledge about the existence of certain types of causal links (and nonexistence of other types) makes the abductive assembly method tractable.

Fourth, the task structure emphasizes that different kinds of methods can be combined: quantitative and qualitative knowledge, heuristic and algorithmic knowledge can be appropriately combined for the accomplishment of a task. For example, if a subtask can be achieved using a known technique, for instance by solving a set of differential equations, that method can be used instead of more traditional AI methods. Since the method that set up this subtask is concerned with the solution, rather than how it was determined, the original task can be implemented using a different kind of method or even a different computational architecture.

Fifth, the generation of new knowledge can itself be viewed as a reasoning task. Hence during knowledge modeling appropriate questions can identify sources of deep knowledge for various methods in the task structure.

Sixth, the task structure outlined can be used to understand the different task-level knowledge-modeling schemes that have been pro-

posed, which were reviewed earlier (see subsection "Background Work in Knowledge Modeling"). KADS, as well as Clancey's heuristic classification have identified extremely general terms or tasks. These tasks can be used to describe almost any method. Hence they can be considered a set of primitive knowledge-modeling terms. Generic task terms are at a higher level of abstraction in the task structure. They are not general enough to be used to describe all methods, but can be used to describe how higher-level tasks such as diagnosis and design can be performed. Generic tasks can, in turn, be described using more primitive terms. This is in fact what has been done throughout the years of research on generic tasks—the large-grained tasks have been repeatedly decomposed into finer-grained tasks.

Finally, the task structure clears up the confusions discussed earlier (see section "Need for Uniform Framework"). These can be addressed as follows:

1. The relation between complex and more primitive generic tasks is, in one sense, relative to the system being described. For example, if a task is implemented by a method that spawns subtasks, we say the subtasks are more primitive, while the higher task is more complex. The concepts are relative because any task, even those lower in the task structure, can potentially be implemented using a complex set of subtasks. In another sense, we can identify tasks appearing as subtasks in a large number of methods as being more primitive or general than tasks appearing in fewer methods. Hence we can say that diagnosis is a less general and more complex task than data abstraction.

2. The variety of knowledge-modeling terms that have been proposed is due to researchers looking at different parts of task structures and having different goals in mind. Some have looked for extremely primitive terms (e.g., Clancey and KADS), while others have tried to identify higher-level terms (e.g., Steels and Generic Tasks). The task structure shows how these terms can be related through tasks, methods and subtasks. There is still a problem with mapping between terms at the same level; however, Clancey's system model-construction perspective (i.e., the view that what KBSs really do is construct models of systems they are reasoning about) provides a scheme to compare these terms by representing each term in a uniform set/graph/operator language [12].

3. Overdetermination and rigidity in methods are avoided by using the task structure because a complete method does not need to be specified (only the subtasks are given and not all of these have to be used to accomplish a task). Furthermore, multiple methods can be used to model domains that do not warrant the selection of a single method for accomplishing a task. Overdetermination and rigidity of implementation can be avoided by dynamically determining methods and subtask sequencing at runtime. Details of how this can be done in the context of generic tasks are given in [18]; but the basic idea is to dynamically determine what to do at each problem-solving step. That is, after each operation is performed the situation is reassessed to determine what can be done next. Knowledge is then brought to bear to select one of these operations.

Knowledge Modeling Is a Task-Specific Enterprise

There have been some attempts to develop a small number of basic terms in which to formulate all the knowledge to be represented in KBSs. We think it premature to discuss representing knowledge in general at this point in our understanding. We can, however, for various types of tasks, develop detailed theories of the methods and knowledge required to implement them and the terms in which such knowledge can be represented. Thus the knowledge-modeling methodology is a cumulative enterprise by researchers around the world: as research in some task (say diagnosis or design) is carried on in various domains around the world, different methods are identified, their knowledge requirements understood, generalizations and commonalities recognized, performance characteristics of the methods are analytically understood, and a task structure which incorporates this collective product of research emerges. Knowledge-modeling for that particular task is then facilitated by this task structure: we know what kinds of knowledge and strategies are needed for the methods and can use the terms of analysis to model the knowledge in the domain.

In this sense, over the last several years significant knowledge has been accumulated for the following tasks: diagnosis, hierarchical design, configuration tasks and some classes of device simulation tasks. We have outlined the task structure for some of these tasks, and shown how the modeling of knowledge is facilitated by this framework.

The usefulness of the task analysis in the form proposed in this article (and the resulting task structure) is not limited to automation of problem-solving. The analysis itself is merely a description of how the task might be decomposed and what kinds of knowledge are needed. It is possible that for one reason or another some of the methods may not be automatable: the needed knowledge may not be available in a computer-processable form, or the method might itself not be sufficiently operationalized. The task analysis still provides the tool for decomposing a task and identifying which subparts of an overall task can be automated. The subtasks for which automatable methods do not exist at a given stage of AI theory-making can be simply directed to a human problem solver with expertise in the subtask. Thus the task structure provides the framework for natural human-machine cooperation. As we understand how to operational-

ize a previously unautomated method and we acquire the knowledge needed for it, that subtask can be given over to the machine. The task structure thus provides a mobile boundary between human and machine in problem-solving.

Acknowledgments

We thank the members of the LAIR and the Division of Medical Informatics for their comments and discussion on this article. B. Ch...rasekaran's work is currently supported by DARPA under AFOSR contract F-49620-89-C-0110. Todd R. Johnson and Jack W. Smith's research is supported by National Heart Lung and Blood Institute grant HL-38776 and National Library of Medicine grant LM-04298.

References

1. Breuker, J. and Wielinga, B. Models of expertise in knowledge acquisition. In *Topics in Expert System Design*, G. Guida, C. Tasso, Eds. Elsevier Science B. V., North-Holland, 1989, pp. 265–295.
2. Brown, D.C. and Chandrasekaran, B. *Design Problem Solving: Knowledge Structures and Control Strategies*. Morgan Kaufmann, San Mateo, Calif., 1989.
3. Bylander, T., Allemang, D., Tanner, M.C. and Josephson, J.R. The computational complexity of abduction. *Artif. Intell. 49*, (1991), 25–60.
4. Bylander, T. and Chandrasekaran, B. Generic Tasks for knowledge-based reasoning: The "right" level of abstraction for knowledge acquisition. *Int. J. Man-Machine Studies 26*, (1987), 231–243.
5. Bylander, T. and Mittal, S. CSRL: A language for classificatory problem solving. *AI VII*, 3 (1986), 66–77.
6. Chandrasekaran, B. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert 1*, 3 (1986), 23–30.
7. Chandrasekaran, B. Design problem solving: A task analysis. *AI Magazine 11*, 4 (1990), 59–71.
8. Chandrasekaran, B. Models versus rules, deep versus compiled content versus form: Some distinctions in knowledge systems research. *IEEE Expert* (Apr. 1991), 75–79.
9. Chandrasekaran, B. and Mittal, S. Conceptual representation of medical knowledge for diagnosis by computer: MDX and related systems. In *Advances in Computers*, M. Yovits, Ed., Academic Press, 1983, 217–293.
10. Chandrasekaran, B., Tanner, M. and Josephson, J. Explaining control strategies in problem solving. *IEEE Expert* (1989), 9–24.
11. Clancey, W.J. Heuristic classification. *Artif. Intell. 27*, 3 (1985), 289–350.
12. Clancey, W.J. Model Construction Operators. *Artif. Intell. 53* (1992), 1–115.
13. Eshelman, L. MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In *Automating Knowledge Acquisition for Expert Systems*, S. Marcus, Ed., Kluwer Academic, 1988, pp. 37–80.
14. Goel, A., Soundararajan, N. and Chandrasekaran, B. Complexity in classificatory reasoning. In *Proceedings of AAAI* (Seattle, Washington, *July 13–18, 1987*) pp. 421–425.
15. Gomez, F. and Chandrasekaran, B. Knowledge organization and distribution for medical diagnosis. *IEEE Trans. Syst., Man and Cybernetics 11*, 1 (1981), 34–42.
16. Gruber, T. and Cohen, P. Design for acquisition: Principles of knowledge system design to facilitate knowledge acquisition. *Int. J. Man-Machine Studies 26*, 2 (1987), 143–159.
17. Johnson, K.A., Johnson, T.R., Smith, J.W., Jr., DeJongh, M., Fischer, O., Amra, N.K. and Bayazitoglu, A. *RedSoar—A system for red blood cell antibody identification*. In *Proceedings of SCAMC 91*, McGraw Hill, Washington D.C., 1991, 664–668.
18. Johnson, T.R. Generic tasks in the problem-space paradigm: Building flexible knowledge systems while using task-level constraints. Ph.D. dissertation, Ohio State University, 1991.
19. Josephson, J., Chandrasekaran, B., Smith, J. and Tanner, M. A mechanism for forming composite explanatory hypotheses. *IEEE Trans. Syst., Man, and Cybernetics 17*, 3 (1987), 445–454.
20. Krishnan, R., Li, X. and Steier, D. Development of a knowledge-based mathematical model formulation system. *Commun. ACM* (Sept. 1992).
21. Laird, J.E., Newell, A. and Rosenbloom, P.S. SOAR: An architecture for general intelligence. *Artif. Intell. 33* (1987), 1–64.
22. Marr, D. *Vision*. W.H. Freeman, New York, N.Y., 1982.
23. McDermott, J. R1: A rule-based configurer of computer systems. *Artif. Intell. 19*, 1 (1982), 39–88.
24. McDermott, J. Preliminary steps toward a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for Expert Systems*, S. Marcus, Ed., Kluwer Academic 1988, pp. 225–256.
25. Mittal, S. and Chandrasekaran, B. Patrec: A knowledge-directed database for a d...gnostic expert system. *Computer 17*, 9 (1984), 51–58.
26. Musen, M.A. *Automated Generation of Model-Based Knowledge-Acquisition Tools*. Morgan Kaufmann, Inc., San Mateo, Calif., 1989.
27. Newell, A. The Knowledge Level. *AI* (Summer 1981), 1–19.
28. Newell, A., Yost, G., Laird, J.E., Rosenbloom, P.S. and Altmann, E. Formulating the problem space computational model. In *Carnegie-Mellon Computer Science: A 25-Year Commemorative*, R.F., Rashid, Ed., ACM Press: Addison-Wesley, Reading, Mass., 1991.
29. Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.
30. Peng, Y. and Reggia, J.A. *Abductive Inference Models for Diagnostic Problem-Solving* Springer-Verlag, New York, N.Y., 1990.
31. Punch, W.F. *A Diagnosis System Using a Task Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning*, Ph.D. dissertation, The Ohio State University, 1989.
32. Reiter, R.A. A theory of diagnosis from first principles. *Artif. Intell. 32* (1987), 57–95.
33. Shortliffe, E.H. *Computer-Based Medical Consultations: MYCIN* Elsevier, New York, N.Y. 1976.
34. Steels, L. Components of expertise. *AI 11*, 2 (1990), 28–49.
35. Wielinga, B.J., Schreiber, A.T. and Breuker, J.A. KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition 4* (1992), 5–53.

CR Categories and Subject Descriptors: D.2.1 **[Software]**: Software Engineering—*requirements/specifications;* D.2.10 **[Software]**: Software Engineering—*design;* I.6.0 **[Computing Methodologies]**: Simulation and Modeling—*general;* I.6.3 **[Computing Methodologies]**: Simulation and Modeling—

applications; K.6.3 [**Computing Milieux**]: Management of Computing and Information Systems—*software management;* K.6.4 [**Computing Milieux**]: Management of Computing and Information Systems—*system management*

General Terms: Design, Methodology

Additional Key Words and Phrases: Analysis, Modeling

About the Authors:

B. CHANDRASEKARAN is professor of computer and information science and director of the Laboratory for AI Research at The Ohio State University. Current research interests include cognitive architectures, knowledge-based reasoning in diagnosis and design, device understanding, and visual reasoning. **Author's Present Address:** Laboratory for AI Research, Department of Computer and Information Science, Ohio State University, Columbus, OH 43210; email: chandra@cis.ohio-state.edu.

TODD R. JOHNSON is an assistant professor with the Department of Pathology, Laboratory for Knowledge-Based Medical Systems at The Ohio State University. Current research interests include flexible problem-solving, modeling the acquisition of expertise, and reasoning with external information.

JACK W. SMITH is associate professor of pathology and computer and information science and director of the Division of Medical Informatics at The Ohio State University. Current research interests include task-specific and cognitive architectures in knowledge intensive domains, abductive problem-solving, and decision support systems in medicine. **Authors' Present Address:** Laboratory for Knowledge-Based Medical Systems, 376 W. 10th Room 571, Columbus, Oh. 43210; email: tj@cis.ohio-state.edu; smith.30@osu.edu

Manuscript Date: June 12, 1993

Form and Content Issues in the Abductive Framework for Recognition[1]

B. Chandrasekaran, John R. Josephson and Susan G. Josephson[2]
Laboratory for AI Research
The Ohio State University
Columbus, OH 43210, USA

Introduction

In an earlier paper, Chandrasekaran and Goel (1988) considered different approaches to the task of classification. The classification problem is one of mapping from observations (data) to pre-enumerated classes. As the classification problem grows complex, the solutions evolve from simple one step numerical mapping to use of intermediate abstractions (symbols) to rules that use relations between abstractions to complex reasoning with knowledge, summarized in the progression:

numbers--> abstractions (symbols) --> relations --> knowledge structures

We compared pattern classification approaches, connectionist networks, syntactic methods and finally knowledge-based classification. The task of classification particularly was deemed important since it is so ubiquitous, playing a role in visual and speech recognition, diagnosis and numerous other tasks of importance. Over the last several years, however, a broader framework called *Abduction* has been gaining attention for many of the same problems that have been traditionally cast in the mold of classification. In this paper, we discuss the structure of the abductive task and relate it to the problem of recognition in general. We will note that classification still has a role as a component in abduction.

Form versus content

One particular idea that didn't seem to come through very clearly in Chandrasekaran and Goel (1988) is the issue of *form versus content.* Often much of the discussion in the field of AI and pattern recognition seems to revolve around whether this mechanism or that (connectionist nets, or logic or symbolic approaches) is the right way to go about building a system to solve some problem (say for visual recognition). We have argued elsewhere (Chandrasekaran, Goel and Allemang, 1989) that, for many purposes, mechanism questions are not the right kind of questions to ask, at least not in the beginning. These questions should be deferred until an understanding of the content of a task is obtained. When the task structure is properly understood, then decisions about what kinds of mechanisms are appropriate for what subtasks can be made. Marr (1982) of course has also made similar arguments, and Newell (1981) is also credited with similar remarks in his discussion on the *Knowledge Level/Symbol Level* distinction. In spite of these well-known analyses, there is still not a suffcient understanding of the importance of content issues in most of the research in the field.

---

[1]For presentation at *Image Processing: Theory and Applications, An International Conference*, Sanremo, Italy, June 16, 1993.

[2]Also with the Columbus College of Art and Design, Columbus, Ohio, USA

In this talk, we will discuss an abductive framework for recognition, and present the discussion largely in terms of the content of the task: what kinds of information are needed, what kinds of subtasks arise the course of performing abduction, and so on. The work that is presented here is elaborated in greater detail in (Josephson and Josephson, 1994).

Abduction

Abduction is a general framework for many important problems in cognition and perception. Abduction has been used to frame the problem of diagnosis, scientific theory formation, natural language understanding, and -- of particular relation to the subject of this conference -- abduction is a more general framework than classification for visual recognition

Abduction or *inference to the best explanation* is a form of inference that goes from data describing something to an explanatory hypothesis that best explains or accounts for the data. Thus abduction is a kind of theory-forming or interpretive inference. We take abduction to be a distinctive kind of inference following this pattern pretty nearly[3]:

D is a collection of data (facts, observations, givens).
H (hypothesis) explains D (would if true, explain D).
No other hypothesis is able to explain D as well as H does.

Therefore, H is probably true.

The core idea is that a body of data provides evidence for a hypothesis that satisfactorily explains or accounts for that data (or at least it provides evidence if the hypothesis is better than explanatory alternatives).

We can see quite easily why visual or auditory recognition fits this pattern of inference. The task is specified by a set of data (pixel intensities, or auditory signal amplitudes at different times), and the goal is to come up with a hypothesis (a description of the scene in terms of objects and their locations, or words that the speaker uttered as he or she produced the sound pattern) that best corresponds to the cause that produced the data. Of course, as a rule, different hypotheses could explain the same data, i.e, the same pixel intensity distribution or temporal sound amplitude distribution could have been caused by different objects or words, so the best that a perceive can do is to come up with the hypothesis that best accounts for the data.

Sometimes a distinction is made between an initial process of coming up with explanatorily useful hypothesis alternatives and a subsequent process of critical evaluation wherein a decision is made as to which explanation is best. Sometimes the term "abduction" has been restricted to the hypothesis-generation phase. We use the term here for the whole process of generation, criticism, and acceptance of explanatory hypotheses. One reason is that, although the explanatory hypotheses in abduction can be simple, more typically they are composite, multipart hypotheses. A scientific theory is

[3] This formulation is largely due to William Lycan.

typically a composite with many separate parts holding together in various ways[4], and so is our understanding of a sentence, and our judgment of a law case, as well as our interpretation of what we see. But no feasible information-processing strategy can afford to explicitly consider all possible combinations of potentially usable theory parts, since the number of combinations grows exponentially with the number of parts available (Josephson and Josephson, 1994). Reasonably-sized problems would take cosmological amounts of time. So, one must typically adopt a strategy which avoids generating all possible explainers. Some pre-screening of theory fragments to remove those that are implausible under the circumstances makes it possible to radically restrict the potential combinations that can be generated, and thus goes a long way towards taming the combinatorial explosion. But such a strategy breaks the clean separation between the process of coming up with explanatory hypotheses, and the process of acceptance, because it mixes a degree of critical evaluation into the process of hypothesis generation. Thus, computationally, it seems best not to neatly separate generation and acceptance. We take "abduction" to include the whole process of generation, criticism, and possible acceptance of explanatory hypotheses.

It seems to us that perception is abduction in layers. We present here a layered-abduction computational model of perception which unifies bottom-up and top-down processing in a single logical and information-processing framework. In this model the processes of interpretation are broken down into discrete layers, where at each layer a best-explanation composite hypothesis is formed of the data presented by the layer or layers below, with the help of information from above. The formation of such a hypothesis is a process of abductive inference. The model treats perception as a kind of frozen or "compiled" deliberation.

Perception as compiled deliberation

There is a long tradition of belief in Philosophy and Psychology that perception relies on some form of inference (Kant, 1787; Bruner, 1957; Gregory, 1987; Fodor, 1983). But this has been typically thought of as some form of deduction, or simple recognition, or feature-based classification, not as abduction. In recent times researchers have occasionally proposed that perception, or at least language understanding, involves some form of abduction or explanation-based inference (Charniak and McDermott, 1985, p.557; Charniak, 1986; Dasigi, 1988; Josephson, 1982, pp. 87-94; Fodor, 1983, pp. 88, 104; Hobbs, Stickel, Martin, et al. 1988). As Peirce, the philosopher who originally coined the term "abduction", wrote, "Abductive inference shades into perceptual judgment without any sharp line of demarcation between them" (Peirce, 1902, p. 304).

Thinking of perception as inferential is not to suppose that perception uses *deliberative* reasoning. The abductive inferences we hypothesize to occur in perception are presumably very efficient, with very little explicit search going on at run time (perception time). Moreover, some of the parts of abductive processing might not be fully realized by actual processing at run time, or they may be done very efficiently with no extraneous support processing. For example alternative hypotheses may be prestored, and simply activated during perception, rather than generated fresh. Similarly,

[4] For example Darden (1991) describes the modularity of genetic theory.

hypothesis interactions, such as degrees of compatibility and incompatibility, may be prestored. This kind of storing of knowledge in just the right form for efficient task-specific processing is just what we mean by "compiled" cognition. It is a kind of thinking well without thinking much.

Cognition might be compiled either as "software" or as "hardware" (or in some intermediate "firmware" form). Moreover, compilation might be done by evolution, or through some sort of learning, or incrementally as needed for perception-time processing. Hardware-compiled structures are presumably not plastic, while learned structures can be subsequently revised (in principle anyway). Thus particular abductive mechanisms might be formed either by evolution, by learning, or at run time. The point of compilation for perception is to avoid computationally expensive run-time search. This can be done by compiling hypothesis fragments and evidential links, as we said. These evidential links may be implemented by currents running along wires, firing rates of connections between neurons, weighted symbol associations, or in some other way; but however they are implemented, they will still really be evidential links (that is, this level of description is not dispensable).

Thus, one way in which perception may plausibly be hypothesized to be very much like deliberation is that the steps and dependencies should make sense logically (abductively). Each piece of processing should be justifiable in ways like ". . . is apparently the only plausible interpretation for this datum," ". . . combine to make this hypothesis better than that one," ". . . was ruled out because . . .," and so on.

Another bridge from perceptual abduction to deliberative reasoning is that there are functionally similar kinds of impasse that can occur during processing. Each such impasse creates a need to fall back on some other form of processing, and provides an opportunity to learn. One such impasse type is where there are no good hypotheses to account for some firmly established data item. In this case (if there is time) a deliberative strategy might be to derive a new hypothesis with the needed properties from background causal knowledge. Apparently this goes on in diagnosis in domains where there are good causal models of the system being diagnosed. Another way to handle this type of impasse is to first capture a description of the data to be accounted for, assume it can be accounted for by some new hypothesis-forming concept *C*, and begin to build up more description of *C* based upon what can be inferred or reasonably conjectured from the context. This additional analysis presumably occurs, not so much immediately at run time in the heat of the original problem solving, but shortly thereafter at "learn time," when run-time information is still available, but urgency has subsided and resources are available for more expensive processing. This learning strategy is neutral between deliberative and perceptual abduction.

A nice example of compiled abduction in perception is that of 3-dimensional vision. The data to be explained are the disparities between the images presented by the two eyes; the explanatory hypotheses are something like overlapping planes, i.e., front-back relationships across edge boundaries.

Perception as Abduction in Layers

Computational models of information processing for things like vision and spoken language understanding have commonly supposed an orderly progression of

layers, beginning near the retina or auditory periphery, where hypotheses are formed about "low-level" features, e.g., edges (in vision) or bursts (in speech perception), and proceeding by stages to higher-level hypotheses. Models intended to be comprehensive often suppose three or more major layers, often with sublayers, and sometimes with parallel channels that separate and combine to support higher-level hypotheses. For example, shading discontinuities and color contrasts may separately support hypotheses about object boundary (Marr, 1982; Lesser, et al., 1975). Recent work on primate vision appears to show the existence of separate channels for information about shading, texture, and color, not all supplying information to the same layers of interpretation (Livingsone and Hubel, 1988). Those easily and naturally are separate channels which then need to converge so as to form a unified hypothesis concerning what is seen.

Another need for layered processing in perception comes from the problem of combining information from the different senses. Combining information from different senses is functionally no different from combining information from different channels within a single sense. The different senses are simply different channels to central, higher "senses." Separate channels within the visual system deliver up the data useful at a certain level to form hypotheses about the locations of 3-d objects; similarly, both sight and hearing can deliver up the data useful for forming hypotheses about object identity. Think of distinctive bird calls, for example, or a lecturer whose identity is uncertain.

One special problem for multi-sense integration is the problem of identifying a "that" delivered up by one sense, with a "that" delivered up by another. Which person is the one that is speaking? Is the same object being seen in the infrared as that being seen in the ultraviolet? Logically, it should be possible for information derived from one sense to help with resolving distinct objects within the other sense. There is actually some evidence that vision can help hearing to separate distinct streams of tones (Massaro, 1987, p. 83) and hear the tone stream as two distinct auditory objects.

One useful representation for multi-sense object perception is that of a "hot map" consisting of overlaid spatial representations, i.e., spatial representations from the different senses brought into "registry" or "correlated" into a single spatial representation. Thus, for example, a robot might bring together separate channels of information from its visual and tactile senses to form a unified spatial representation of its immediate surroundings. Such a map should be maintained continually, and updated and revised as new information arrives and is interpreted. This hot map, with its symbols on it, can be seen as the resulting composite hypothesis formed by a process of layered abductive interpretation.

In vision most of the processing of information is presumably bottom-up, from information produced by the sensory organ, through intermediate representations, to the abstract cognitive categories that are used for reasoning. Yet top-down processing is presumably also significant, as higher-level information imposes biases, and helps with identification and disambiguation. Vision can thus be thought of as a layered interpretation task wherein the output from one layer becomes data to be interpreted at the next. In image understanding the layers are something like this:
Retinal Activation --> segmentation, edges, regions --> 2-1/2D overlapping planes, occlusions --> grouping --> object ID, scene analysis

Layers may be fixed in advance or formed at run time. Output from one layer becomes data to be interpreted at the next layer. Layered interpretation models for non-

perceptual interpretive processes make sense too. For example, medical diagnosis can be thought of as an inference that typically proceeds from symptoms, to pathological states, to diseases, to disease processes to etiologies. In speech comprehension, for example, the layers might be acoustic signals to phonetics to grammar and clauses etc. to semantics. Perception, speech comprehension and medical diagnosis all have a similar pattern of layers of abductive processing. Similar to perception, medical diagnosis is presumably mostly bottom up, but with a significant amount of top-down processing serving similar functions.

It is reasonable to expect that perceptual processes have been optimized over evolutionary time (become efficient, not necessarily optimal), and that the specific layers and their hypotheses, especially at lower levels, have been compiled into special-purpose mechanisms. Within the life span of a single organism, perceptual learning provides additional opportunities for compilation and optimization. Nevertheless, it seems that at each layer of interpretation the abstract information-processing task is the same: that of forming a coherent, composite best explanation of the data from the previous layer or layers. That is, the task is abduction, and in particular, abduction requiring the formation of composite hypotheses.

If the information processing that occurs in the various layers and senses is functionally similar, then perhaps their mechanisms are similar too at a certain level of description. Thus we are led to hypothesize that the information-processing mechanisms that occur in vision, hearing, understanding spoken language, and in interpreting information from other senses (natural and robotic), are all variations, incomplete realizations, or compilations (domain-specific optimizations) of one basic computational mechanism. Thus we propose what we may call *the layered-abduction model of perception*. What is new in this model is the specific hypothesis that perception uses abductive inferences, occurring in layers, together with a specific computational model of abductive processing.

Task Structure Model of Abduction in Layers

A task structure (Chandrasekaran, 1990; Chandrasekaran and Johnson, 1993) is a representation of a task in terms of the methods that are applicable for it in the domain and the conditions under which each method is applicable. A task analysis is a functional decomposition of an information processing task. Such an analysis is intended to answer questions about how it is possible to accomplish the task, especially, how the task can be feasibly accomplished. Thus, the task is divided into subtasks and those subtasks are divided into sub-subtasks, and so on. The subtasks are those things which are required to accomplish the task, and the sub-subtasks are those things which are required to accomplish the subtasks.

A task structure analysis identifies domain-specific and domain independent aspects of the tasks and methods for their accomplishment. The task can be feasibly accomplished if a decomposition can be found such that each subtask is feasible and can be combined with the other subtasks by a feasible method.

For each task (subtask, sub-subtask, etc.) there are one or more possible methods that can be used to accomplish it. Methods are ways to accomplish some task, like using an abacus or using your fingers or using a calculator are all alternative methods for performing the task of addition. Each method is itself specified in terms of how it uses

knowledge and inference to achieve its goals, and in terms of what subgoals (subtasks) it sets up and requires to be achieved before it can succeed. This kind of decomposition can be done recursively until methods which achieve subgoals but which do not set up additional subgoals of their own are reached. Alternative methods for accomplishing the task may make use of a common subtask, and so on recursively.

A task can have one or more methods associated with accomplishing it. Each of the methods is characterized by forms of knowledge and inference that are necessary for carrying out the method and by additional subtasks that will need to be achieved in order to complete the application of the method for that task. A method can be a procedure where the sequencing of steps is all prespecified, but it can be more abstract; in Newell's problem space terminology [Newell, 1980], it can be a search in a problem space. (In fact, such methods are the ones that are interesting from an AI point of view.) For example, the problem of *classification* has a method called *hierarchical classification*, which consists of exploring the classification hypotheses organized as a hierarchy. This calls for knowledge in the form of *hierarchies* and inference methods which are variations of, and include as a default strategy, *top-down explorations* of the hierarchy. This method has subtasks in the form of *evaluating* the evidence for or against a hypothesis so that it can be established or rejected. This subtask similarly can have many methods associated with it, each of which is characterized by its own knowledge and inference requirements.

We can analyze the task of abduction to have two main subtasks:

- subtask 1) *generating elementary hypotheses*,
- subtask 2) *synthesizing composite explanations*.

The subtask, *generating elementary hypotheses*, has two sub-subtasks:

- sub-subtask 1.1) *evoking*
- sub-subtask 1.2) *instantiating elementary hypotheses*

The sub-subtask 1.2., *instantiate elementary hypothesis*, has two sub-sub-subtasks:

- sub-sub-subtask 1.2.1) *scoring the elementary hypotheses*
- sub-sub-subtask 1.2.2) *determining explanatory coverage*.

This decomposition is very general. The typical abductive answer is a composite, and somehow, explicitly or implicitly, there must be some method of choosing which elementary hypothesis to consider, some way of making them specific to the case, and some way of accepting and combining. Given this task structure, the information processing at each layer is decomposed into three functionally distinct types of activity: *evocation of hypotheses, instantiation of hypotheses*, which are the two sub-subtasks making up the first subtask, and *composition of hypotheses*, which makes up the second subtask of the overall abductive task.

Evocation (sub-subtask 1.1) can occur bottom-up, a hypothesis being stimulated for considerations which are cued by the data presented at a layer below. That is, the presence of a certain finding suggests that certain hypotheses are appropriate to consider. More than one hypothesis may be suggested by a given datum. Evocation can also occur top-down, either as the result of priming (an expectation from a level above), or as a consequence of data-seeking activity from above, which can arise from the need for evaluation. As when you what to decide whether the person coming towards you is really Jane, and so you look for Jane's characteristic mole on the left cheek. Evocations can generally be performed in parallel, and need not be synchronized.

Instantiation (sub-subtask 1.2) occurs when each stimulated hypothesis is independently scored for confidence (sub-sub-subtask 1.2.1), and a determination is made about what part or aspect of the data is accounted for by the hypothesis (sub-sub-subtask 1.2.2). Instantiation is in general top-down.[5] During instantiation data may be sought that was not part of the original stimulus for evoking a hypothesis. Each hypothesis is given a confidence value on some scale, which can be taken to be a "local match" or *prima facie* likelihood, a likelihood of being true based only on consideration of the match between the hypothesis and the data, with no consideration of interactions between potentially rival or otherwise related hypotheses. Typically, many evoked hypotheses will get very low scores, and can be tentatively eliminated from further consideration. The data that are accounted for by a hypothesis may or may not be identical to the data on the basis of which the hypothesis was scored, or the data that did the evoking.

In the course of instantiation the hypothesis set may be expanded by including subtypes and supertypes of high-confidence hypotheses, if the space of potential hypotheses is organized hierarchically by level of specificity. Instantiation is done most efficiently when it is based on matching against prestored patterns of features, but slower processes of instantiation are also possible whereby the features to match are generated at run time.

The result of a wave of instantiation activity is a set of hypotheses, each with some measure of confidence, and each offering to account for some portion of the data. Since within a particular wave of instantiation, hypotheses are considered independently of each other, the process of instantiation can go on in parallel.

Composition (subtask 2) occurs when the instantiated hypotheses interact with each other and (under good conditions) a coherent best interpretation emerges. At the beginning many hypotheses will probably have intermediate scores, representing hypotheses that can neither be taken as practically certain, nor as being of such a low confidence as to be ignorable. So knowledge of interactions between the hypotheses is brought to bear to reduce the degree of uncertainty, increasing confidence in some of them, and decreasing confidence in others. (See Josephson and Josephson, 1994, for details of strategies.)

Top-down information flow

In the layered-abduction computational model composite hypotheses are formed in several places at once in a coordinated fashion. Each locus of hypothesis formation we call an *agora*, after the market place where the ancient Greeks would gather for dialog and debate. The idea is that an agora is a place where hypotheses of a certain type gather, and contend, and where under good conditions a consensus hypothesis emerges. In typical cases the emerging hypothesis will be a composite, coherent in itself, and with different sub-hypotheses accounting for different portions of the data. For example in vision the "edge agora" is the presumed location where edge hypotheses are formed and

[5] Under certain circumstances it is good enough just to score on the basis of "voting" by the stimulating data from below, and then no top-down processing need occur, at least for scoring. Though this strategy is less than logically ideal, it is computationally less expensive, at least in the short run. This represents a kind of compilation where accuracy is traded for quickness.

accepted, each specific edge hypothesis accounting for certain specific data from lower-level agoras.

The agoras are organized in an information-flow network with a clear sense of direction defining the difference between bottom-up and top-down flow. Bottom-up is from data to interpretation. The main output from an agora is "upwards," the data to be interpreted by "higher" agoras. Another possible output is "downwards," for example expectation might influence the consideration and evaluation of hypotheses at lower agoras. Thus the relationship between "data" and "explanation" is a relative one, and an explanation accepted at one agora becomes data to be explained at the next. Though we sometimes use the word "layer" to describe an agora or contiguous cluster of them, we do not suppose that the agoras are all neatly lined up. The paths may be branching and joining (but no cycles are permitted). The basic strategy is to try to solve the overall abduction problem at a given agora by solving a sufficient number of smaller and easier abductive sub-problems at that agora.

Downward-flowing information processing between layers can occur in at least four ways. One is that the data-seeking needs of hypothesis evaluation or discrimination can provoke instantiation (top-down evocation and evaluation of a hypothesis). Another is that expectations based on firmly established hypotheses at one layer can "prime" certain data items (i.e., evoke consideration of them and bias their score upwards). A third way is that hypotheses that are uninterpretable as data at the higher level (no explanation can be found) can be "doubted," and reconsideration of them provoked (also reconsideration of any higher-level hypotheses whose confidence depended on the questionable datum). Finally, data pairs may be jointly uninterpretable, as for example in vision when the image has two mutually incompatible interpretations (to some degree of strength), and reconsideration can be provoked from above. In these ways higher-level interpretations can exert a strong influence on the formation of hypotheses at lower levels, and layer-layer harmony is a two-sided negotiation.

We may summarize the control strategy for hypothesis-composition by saying that it employs multi-level and multiple intra-level island-driven processing. Islands of relative certainty are seeded by local abductions and propagate laterally (incompatibilities, positive associations), downwards (expectations), and upwards (accepted hypotheses become data to be accounted for). Processing occurs concurrently and in a distributed fashion. Higher levels provide *soft* constraints through the impact of expectations on hypothesis evocation and scoring, but this does not strictly limit the hypotheses that may be accepted at lower levels.

Conclusion

Notice the level of description we have given here. This architecture is a high-level architecture and is potentially compatible with a number of different implementation architectures, such a neural nets or traditional symbolic programming. It might be implemented as an "algorithmic computer",that is, an instruction follower, or as a "connectionist computer," whose primitive processing elements work by propagating activations. We have described the functional and semantic significance of various actions of the machine, and the flow of control, but not precisely how these actions are implemented on some underlying machine. That is, once we view the problem in a task

structure way, we have a content-driven perspective. The layered abduction view does not commit us to using connectionism, or symbolic processing. It is a theory of the information processing task, that is, a computational theory which is a description of the input output relations, and by saying it occurs in layers and how it occurs at each level in terms of subtasks, etc, we are describing the strategy for the task. This task structure level is above the level of algorithms and data structures, or nodes and weights, which in turn are a level above the level of implementation. In this sense the task structure analysis is a content theory.

A task structure has an enormous amount of leverage in directing knowledge acquisition and system building, since the knowledge and inference requirements for the methods can be explicitly identified. The task-oriented view has significant potential to aid learning. In addition to the knowledge-type vocabulary that it provides for each task, its ability to generate explanations at the right level can give significant leverage for learning (Chandrasekaran,1986, 1987, Bylander and Chandrasekaran, 1987).

Acknowledgments

B. Chandrasekaran thanks Prof. Gianni Vernazza for his invitation to give a talk at the conference. We also acknowledge the support of DARPA under AFOSR Contract F-49620-89-C-0110, and of NEC Corporation for research reported here.

References

Bruner, J. S., 1957, "On perceptual Readiness," *Psychological Review*, 64:2, pp.123-152.

Bylander, T. and Chandrasekaran, B., 1987, "Generic tasks for knowledge based reasoning; the "right" level of abstraction for knowledge acquisition," *International Journal of Man-Machine Studies*, 26, 231-243.

Chandrasekaran, B., 1986, "Generic tasks in knowledge-based reasoning: High level building blocks for expert system design" *IEEE Expert*, 1, pp. 23-30.

Chandrasekaran, B., 1987, "Towards a functional architecture for intelligence based on generic information processing tasks" Invited talk, *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI

Chandrasekaran, B. and Goel, A., 1988, "From numbers to symbols to knowledge structures: Pattern Recognition and Artificial Intelligence perspectives on the classification task," *IEEE Trans. Systems, Man and Cybernetics*, Vol. 18, No. 3, May/June 1988, pg. 415-424.

Chandrasekaran, B., Goel, A. and Allemang, D., 1989, "Connectionism and information processing abstractions: the message still counts more than the medium," *AI magazine*, 9:4, pp. 24-34, 1989.

Chandrasekaran, B., Tanner, M., and Josephson, J.,1989,"Explaining control strategies in problem solving," *IEEE Expert*, 4, pp. 9-24.

Chandrasekaran, B., 1990, "Design problem solving: A task analysis," *AI Magazine*, 9:1, pp. 9-17.

Chandrasekaran, B., and Johnson, T. R., 1993, "Generic tasks and task structures: History, critique and new directions," in J-M. David, J-P. Krivine and R. Simmons, Editors, *Second Generation Expert Systems*, Berlin: Springer Verlag.

Charniak, E., and McDermott, D.,1985, *Introduction to Artificial Intelligence*, Reading, MA: Addison-Wesley. p.557;

Charniak, E., 1986, "A Neat Theory of Marker Passing," in *Proceedings of AAAI-86*, Volume 1, pp. 584-588, AAAI, Morgan Kaufmann, August.

Dasigi, V. R., 1988, *Word Sense Disambiguation in Descriptive Text Interpretation: A dual-route Parsimonious Covering Model*. PhD thesis, University of Maryland, College Park.

Fodor, J., 1983, *The Modularity of Mind*, Bradford Book, MIT press.

Gregory, R. L., 1987, "Perceptions as Hypotheses," in Richard L. Gregory , editor, *The Oxford Companion to the Mind*, , Oxford University Press, pp. 608-611.

Hobbs, J. R., Stickel, M., Martin, P., and Edwards,D.,1988, "Interpretation as Abduction" in *Proc. 26th Annual Meeting of the Assoc. for Computational Linguistics*, Buffalo, NY, pp. 95-103.

Josephson, J. R.,1982, *Explanation and Induction* , PhD thesis, The Ohio State University, Columbus, Ohio. pp. 87-94.

Josephson, J. and Josephson,S., 1994, Editors, *Abductive Inference: Computation, Philosophy, Technology* , Cambridge University Press, 1994, (forthcoming)

Kant, I. ,1787, *Critique of Pure Reason*, St Martin Press, (1968), New York, tr. Norman Kemp Smith.

Laird, J.E. , Newell, A. and Rosenbloom, P.S., 1987," SOAR: An architecture for general intelligence," *Artificial Intelligence*, 33, pp 1-64.

Lesser, V.R., Fennell, L.D., Erman, and Reddy, R.D., 1975, "The Hearsay II speech understanding system" *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-23, pp.11-24.

Livingstone , M. and Hubel, D.,1988, "Segregation of form, color, movement, and depth: Anatomy, physiology, and perception," *Science*, 240, May.

Marcus, S. and McCermott, J. 1989. "Salt:A knowledge acquisition tool for propose-and-revise systems," *Artificial intelligence*, 39, pp. 1-37.

Marr, D.,1982, *Vision*, W.H. Freeman and Company, New York.

Massaro, D.W.,1987, *Speech Perception by Ear and Eye: A Paradigm for Psychological Inquiry*, Lawrence Erlbaum Associates, New Jersey, p. 83.

Newell, A.,1980, "Reasoning, problem solving and decision processes: The problem space as a fundamental category" in R. Nickerson, Editor, *Attention and performance*, VIII, Lawrence Erlbaum Associates, New Jersey, pp. 693-718.

Newell, A.,1981, "The Knowledge Level," *AI Magazine*, 2:2, pp. 1-64.

Peirce, C.S. 1902, "Perceptual Judgements" reprinted in Buchler, J., editor, *Philosophical Writings of Peirce*, (1955) Dover, pp. 302-305.

To appear in "Second Generation Expert Systems", edited by Jean-Marc David, Jean-Paul Krivine and Reid Simmons, Springer Verlag.

Explanation Using Task Structure and Domain Functional Models

Michael C. Tanner[1], Anne M. Keuneke[2], and B. Chandrasekaran[3]

[1] Computer Science Department, George Mason University, Fairfax VA 22030, USA
[2] Computer Science Department, California State University, Chico CA 95929, USA
[3] Department of Computer and Information Science, The Ohio State University, Columbus OH 43210, USA

Abstract. In this paper we present some of the work and ideas developed at the Ohio State Laboratory for AI Research on explaining the behavior of knowledge systems. The first part of the paper presents an analysis of the explanation problem and the aspects of it that we have concentrated on (briefly, we are concerned more with the form and content of the representations than the explanation form or presentation). Then we describe a generic task-based approach to explanation, including relating the explanation to the logical structure of the task. Finally, we show how causal models of a domain can be used to give explanations of diagnostic decisions.

1 Aspects of Explanation

As described by Chandrasekaran, Tanner, and Josephson [8], we can separate the explanation generation problem in knowledge systems into three top-level functions: generating the content, being responsive, and interacting with human users.

Generating an explanation's basic content. Given user queries about a system's decisions, we need to generate an information structure containing the elements needed for an explanation.

Shaping explanations to match user knowledge. It may not be necessary to communicate all the available explanation content to users. Systems apply knowledge of user goals, state of knowledge, and the dialog structure to filter, shape, and organize the output of the above *content* process so that explanations respond to user needs.

Interacting with users. The two preceding functions produce all the information needed conceptually and logically for the required explanation. However, presentation issues remain; specifically, how an appropriate human-computer interface effectively displays and presents information to users.

---

* Parts of this paper appeared in B. Chandrasekaran, M. C. Tanner, and J. R. Josephson, "Explaining control strategies in problem solving," *IEEE Expert*, 4(1), pp. 9–24, 1989, and M. C. Tanner and A. M. Keuneke, "The role of the task structure and domain functional models," *IEEE Expert*, 6(3),1991, pp. 50-57. Reused by permission of IEEE Computer Society.

If explanation content is inadequate or inappropriate – no matter how good theories for responsiveness and interface functions are – then correspondingly poor explanations will be presented. Thus, generating the correct explanation content is the central problem in explanation generation. We can break this down into the following types:

Step explanation. Relating portions of the data in a particular case to the knowledge for making specific decisions or choices, i.e., explaining the steps in the solution process.

Strategic explanation. Relating decisions to follow particular lines of reasoning to the problem solver's goals.

Task explanation. Relating the system's actions and conclusions to the goals of the task it performs.

Knowledge justification. Relating conclusions and problem-solving knowledge to other domain knowledge, possibly showing how they were obtained.

These four kinds of explanation are related to the act, or process, of solving problems. A knowledge system might be asked questions about many other relevant things, including requests for definition of terms and exam-like questions that test the system's knowledge. Answers to these questions may or may not be explanations, as such, but a knowledge system should still be able to produce them. All of these kinds of explanation correspond to structures that must be examined when constructing explanations, even though some of the structures may not be needed to solve problems in the system's domain.

Often explanations are produced by introspection, i.e., a program examines *its own* knowledge and problem-solving memory to explain itself. Step and strategic explanations are most often done this way. But sometimes explanations are concocted, i.e., they do not relate to how the decision was actually made, but independently make decisions plausible. Constructing such *post facto* justifications or explanations is necessary when problem solvers have no access to their own problem solving records, or when the information contained in those records is incomprehensible to users. The explanation may argue convincingly that the answer is correct without actually referring to the derivation process, just as mathematical proofs persuade without representing the process by which mathematicians derive theorems. Task explanations and knowledge justifications are often done this way. Generating explanations of this sort is an interesting problem solving process in its own right [23, 39].

2 Tasks, Methods and Explanations

In this section we give a brief outline of the notion of a task analysis as developed by Chandrasekaran [3]. Explaining a knowledge system's solutions requires, among other things, showing on one hand how the logical requirements of the task were satisfied by the solution and, on the other hand, showing how the method adopted (the strategy) achieved the task in the problem-solving instance. In principle there may be more than one method for a task. Most knowledge systems have "hard-wired" specific methods for the tasks. Thus Mycin can

be understood as solving the diagnostic task by the method of heuristic classification, which in turn performs the subtasks of data analysis, heuristic match, and refinement [9]. In the generic task (GT) framework, developed at Ohio State, we have identified a number of task-method combinations that can be used as building blocks for more complex tasks. Thus, for example, the task of diagnosis is associated with a generic method called abductive assembly, which in turn sets up a subtask of hypothesis generation. In our GT work, a generic method called hierarchical classification is proposed for exploring certain types of hypothesis spaces. This in turn sets up subtasks for evaluating hypotheses in the hierarchy for which a generic method called hierarchical evidence abstraction is proposed. What we have called GTs are in fact task-method combinations. The method is particularly appropriate to the task because it is commonly used in many domains for that task and it gives significant computational advantages. Two of the GTs we have identified are Hierarchical Classification and Design by Plan Selection and Refinement.[4]

Hierarchical Classification

Task: If a hypothesis hierarchy is available, generate hypotheses that match the data describing a situation.

Method: For each hypothesis, set up a subtask to establish or reject it. If it is established, test its successors. If it is rejected, it and its successors are rejected. The top-down control strategy, called Establish-Refine, can be varied under specific conditions. Bylander and Mittal [2] elaborate on this simplified account.

Design by Plan Selection and Refinement

Task: Design an object that satisfies certain specifications.

Method: Design is separated into a hierarchy of subdesign problems, mirroring the object's component structure. For each node in the hierarchy, there are plans for making commitments for some component parameters. Each component is designed by choosing a plan, based on some specifications, which instantiates some design parts and designs further subcomponents to fill in other parts. We describe this task in more detail in Sect. 3, but Brown and Chandrasekaran [1] is the definitive reference on this topic.

Each GT method is explicitly supported by a high-level language that aids knowledge system development by giving the knowledge engineer access to tools that work closer to the problem level, not the rule or frame level. However, it may be necessary to divide non-trivial problems into subproblems such that each matches some GT. This way of building complex knowledge systems also means that knowledge engineering environments should provide a tool set rather than a single tool. Consequently, some of our recent work has concentrated on developing the Generic Task Toolset [21].

[4] The following description differs from descriptions in earlier papers [4], since we have separated the task and the method explicitly.

In using the GT theory for explanation we need to show how the method-specific high-level language helps in explicitly and directly generating explanations of strategy at the right level. This is what our early work involved, and is described in Sect. 3. In general, however, we also need to relate the logical structure of the task to the strategy employed. Issues involved in this are discussed in Sect. 4. Then in Sects. 5 and 6 we describe work on justifying problem-solving knowledge by reference to the more general knowledge on which it is based.

3 Generic Tasks and Explanation – An Example

GTs make strategic explanation possible for systems built using this theory [8]. Additionally, any explanation facility should be able to explain the steps a problem solver takes in reaching a solution. In this section we describe MPA (a Mission Planning Assistant) [18], a GT program capable of explaining its problem-solving steps and its strategy. We transferred the techniques developed on this system to the Generic Task Toolset [21] so that any system built using those tools could explain its steps and strategy.

MPA is a GT implementation of Knobs [13], a system that plans a particular kind of Air Force mission.[5] The type of planning required can be viewed as design, i.e., designing the plan. So we used the GT of Design by Plan Selection and Refinement, and implemented MPA using the generic task tool DSPL (Design Specialists and Plans Language) [1].

3.1 Overview of DSPL

A design problem solver in DSPL is a hierarchy of specialists, each responsible for a specific design portion (see Fig. 1). Specialists higher up in the hierarchy deal with the more-general aspects of devices being designed, while specialists lower in the hierarchy design more-specific subportions. The organization of the specialists, and the specific content of each, is intended to capture design expertise in the problem domain.

Each specialist contains design knowledge necessary to accomplish a portion of the design (see Fig. 2). Each specialist has several types of knowledge but, for simplicity, we will describe only three. First, explicit design plans in each specialist encode sequences of possible actions to successfully complete that specialist's task. The plans consist of design steps, each of which chooses a value for one parameter of the design. Second, specialists have design plan sponsors associated with each plan to determine that plan's appropriateness in the runtime context. And third, each specialist has a design plan selector to examine runtime judgments of sponsors and to determine the plan most appropriate to the current problem context.

In a DSPL system, control proceeds from the topmost design specialist to the lowest. Each specialist selects a plan appropriate to the problem's requirements.

[5] MPA actually implements a very simplified version of the problem.

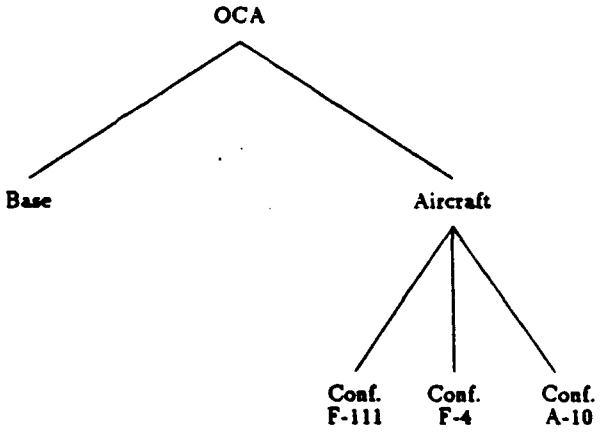


Fig. 1. Organization of MPA, a DSPL Problem Solver

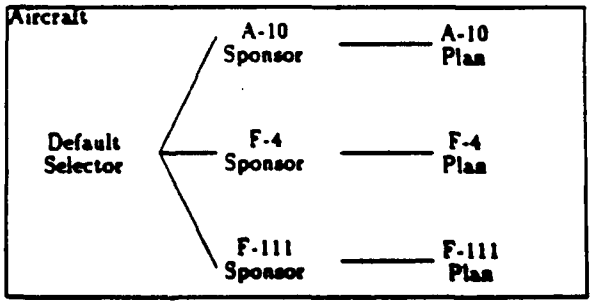


Fig. 2. Inside a DSPL Specialist

The system executes plans by performing design actions that the plan specifies (which may include computing and assigning specific values to device attributes, checking constraints, or invoking subspecialists to complete another portion of the design).

3.2 Description of MPA

In MPA the top-level specialist is OCA (for Offensive Counter Air, the type of mission MPA works on), which has design plans for producing the mission plan. Subcomponents of the mission are the Base and the Aircraft (see Fig. 1). The Base specialist chooses the air base (actually a list of bases) using the requirements of the mission. The Aircraft specialist chooses an aircraft type, and then configures the aircraft for the mission using subspecialists.

As an example of a specialist, consider Aircraft (shown in Fig. 2). It contains a selector, in this case it is the default selector built into DSPL. The default selector simply chooses the best plan, according to ratings assigned by the sponsors, and if there are no good plans it fails.[6] Aircraft also contains three sponsors, one for each of its plans. It has a plan for each aircraft type (A-10, F-4, and F-111).

The DSPL code for Aircraft's A-10 Plan is given in Fig. 3. MPA decides whether A-10 is the appropriate aircraft type for the mission using its sponsor-selector mechanism described above. If A-10 is appropriate, this plan is executed. The BODY contains a list of the steps in the plan. It first notes the aircraft type, then chooses a squadron. The base is determined from the chosen squadron and then the range to the target is computed. Finally, the aircraft is configured for the mission (bomb and fuel load) by calling the subspecialist Configure-A-10.

```
(PLAN
   (NAME A-10)
   (SPONSOR A-10-Sponsor)
   (PURPOSE ''considering the feasibility of an A-10 for the mission'')
   (ACHIEVED ''chose an A-10 for the mission'')
   (BODY
     AssignAircraftType
     ChooseSquadron
     AssignBase
     GetRange
     (DESIGN Configure-A-10)))
```

Fig. 3. MPA's A-10 Plan

One of the steps, ChooseSquadron, is shown in Fig. 4. Steps in DSPL set the value of a single design attribute, so the step first identifies the attribute it sets

[6] DSPL contains a mechanism, not described here, for dealing with failures. Brown and Chandrasekaran [1] provide the details.

(SQUAD). The DSPL functions KB-FETCH and KB-STORE fetch and store attribute values of the design. The KNOWN section of the body is a facility for defining local variables for known attributes. REPLY contains the main work of the step. In this case the REPLY simply stores an attribute value but, in general, it could do more work to decide on the value. The Lisp function squad-select implements the expert's method of choosing a squadron given the aircraft type and bases available. Since this is done in a Lisp function, it will not be easily explainable. A better implementation of MPA would include this decision process in DSPL, rather than in Lisp.

```
(STEP
   (NAME ChooseSquadron)
   (ATTRIBUTE-NAME SQUAD)
   (PURPOSE ''selecting a squadron for the mission'')
   (ACHIEVED ''selection of & as the squadron for the mission'')
   (BODY
     (KNOWN
       plane      (KB-FETCH AIRCRAFT)
       base-list  (KB-FETCH BASELIST))
     (REPLY
       (KB-STORE SQUAD (squad-select plane base-list))
```

Fig. 4. MPA's ChooseSquadron Step

In the end, MPA produces a list of attributes for the mission with the values it decided upon. For example:

Aircraft Type	A-10
Number of Aircraft	4
Squadron	118TFW
Airbase	Sembach
⋮	⋮

This list is actually a menu, from which users can select any value and ask MPA to explain how that value was decided.

3.3 Explanation in MPA

We implemented explanation in MPA on the organizing principle that the agent that makes a decision is responsible for explaining it. For our purposes, we consider the things we have described - specialists, selectors, sponsors, plans, and steps (Figs. 2–4) - to be problem-solving agents in DSPL. The current implementation of MPA contains nearly 100 of these agents, which call upon each other during the problem-solving process. All of these agents have well-defined roles,

so the system can explain an agent's decisions in terms of the goals of its calling agent, the agent's own role in the pursuit of those goals, and the roles of other agents it called upon. To do this we added slots called **PURPOSE** and **ACHIEVED** to the agent definitions in DSPL to hold text strings for describing the agents' goals. Then to explain how MPA decided on a particular attribute value, the explanation module puts these strings together in an order that depends on the runtime context in which the decision was made. Given such an explanation users can select any of the other agents and ask for further elaboration from them.

Suppose a user selected the value "118TFW" of the attribute "Unit". The only question users can ask MPA, the only explanation it can give, is a form of "How was it decided?" Thus, the user's selection in this case implicitly asks, "How was it decided that the Unit should be 118TFW?" The explanation is given in Fig. 5. This decision was made by the **ChooseSquadron** step so the explanation comes from that agent. The explanation first gives the purpose of the calling agent (shown in italics), which comes from the A-10 plan in this case. Then it gives the values of the local variables. Finally, it gives the value it chose for its attribute.

The context of *considering the feasibility of an A-10 for the mission* determined that:

- plane was **A-10**
- base-list was **Ramstein, Bitburg, Sembach**

So **118TFW** was an appropriate choice for Unit.

Fig. 5. Explanation for a Step

This explanation may be unsatisfying. A better explanation in this case might be:

> Assuming that we are to use A-10s and that the only bases available are Ramstein, Bitburg, and Sembach, then 118TFW is the only unit that flies A-10s out of these bases.

Some of the difference between this and Fig. 5 is the quality of the English text. The only content difference, and content has been our focus, is in connecting the assumptions (values of local variables) to the final decision. MPA could do this better if the final decision were made using DSPL rather than the Lisp function **squad-select**. A slightly improved version of the explanation would appear as in Fig. 6. Because the explanation module is essentially just translating DSPL code into text, the quality of the programming affects the quality of the explanation. This is a little bit undesirable but also unavoidable in a system that has to explain itself using only its own problem-solving knowledge.

Users can select any of the local variables given in the explanation (i.e., **plane** and **base-list**) for further elaboration. For example, to find out why **plane** is A-10. This would result in getting an explanation from another step,

The context of *considering the feasibility of an A-10 for the mission* determined that:

- plane was A-10
- base-list was Ramstein, Bitburg, Sembach
- units-with-A10 was 118TFW

So 118TFW was an appropriate choice for Unit.

Fig. 6. Improved Explanation for a Step

since attribute values are determined by steps. Or users can select the calling agent for further explanation. This would result in an explanation from the A-10 plan, shown in Fig. 7. As with the step explanation, the context comes from the calling agent, the Aircraft specialist here. The bulleted items are the purposes from the called agents. Additionally, the explanation shows where the agent was in its procedure. In this explanation, since the user arrived here from the ChooseSquadron step, the plan had completed the AssignAircraftType step, was in the process of doing the ChooseSquadron step, and had yet to do the AssignBase and GetRange steps and complete the configuration.

In the context of *selecting an appropriate aircraft for the mission* I:

- Assigned A-10 as the aircraft type.

I was in the process of:

- Selecting a squadron for the mission.

and was about to do to following:

- Select a base for the mission.
- Determine the range for the mission.
- Choose a configuration for the A-10 on this mission.

Fig. 7. Explanation for a Plan

The explanations shown here are generated from explanation templates. Each agent type has a standard representation form from which we derived its explanation template. A simplified version of the standard form for plans is shown in Fig. 8 (the simplification is that in addition to steps, plans can contain DESIGN calls to subspecialists as in Fig. 3). Figure 9 shows a correspondingly simplified explanation template for plans, assuming that it is entered from step i. Thus, a plan's explanations are put together out of the goals of its calling specialist and the goals of the steps it calls.

Putting explanations together out of "canned" text, the way MPA does, is not a very sophisticated method of text generation. However, the important point here is that the roles of the various agents - specialists, plans, steps, etc. - and their relationships define the kinds of things that can be said and how these things go together to make sensible explanations. These roles and relationships

```
(PLAN
    (NAME (plan name))
    (SPONSOR (sponsor name))
    (PURPOSE (purpose string))
    (ACHIEVED (achieved string))
    (BODY
        (step 1)
        ⋮
        (step i)
        ⋮
        (step n)))
```

Fig. 8. Standard Plan Representation

In the context of (purpose of containing specialist) I:

- (achieved string from step 1)
- ⋮
- (achieved string from step $i-1$)

I was in the process of:

- (purpose string from step i)

and was about to do to following:

- (purpose string from step $i+1$)
- ⋮
- (purpose string from step n)

Fig. 9. Plan Explanation Template

are defined by the GT, in this case Design by Plan Selection and Refinement. We have more work to do on developing a taxonomy of PURPOSEs for the various agents, and then showing how to use the taxonomy for explaining. However, our aim for MPA was to demonstrate that GT programs provide the structures needed to generate explanations of strategy and steps.

4 Explanation Based on the Logical Structure of a Task

GTs combine tasks with appropriate methods, which enables explanations to show how strategic elements combine to achieve the task's major goals. However, as described by Chandrasekaran [3] (see Sect. 3), for any task there are many possible methods. To properly explain how a program's knowledge, strategy, behavior, and conclusions relate to its problem-solving task, we need to

separate the task's requirements from those of the methods that perform it. For example, one diagnostic goal is to find a disease that explains the symptoms. One method would produce explanatory hypotheses using disease hierarchies, another would produce them using causal models. Each method imposes its own requirements and has a distinctive behavior, but both serve the same diagnostic subgoal – generating explanatory hypotheses. An explanation should relate their behavior to their subgoal in spite of the detailed differences between them. So it is important to identify tasks' logical structure, independent of particular solution methods, to be used in designing explanation components for systems that perform them. In this section we describe Tanner's work on task explanation in diagnosis [36].

4.1 The Logical Structure of Diagnosis

Diagnosis is usually considered an abduction problem [12, 17, 20, 28, 29, 30]. That is, the task is to find a disease, or set of diseases, that best explains the symptoms. Accordingly, a diagnostic conclusion is supported, perhaps implicitly, by the following argument:

- There is a principal complaint, i.e., a collection of symptoms that sets the diagnostic problem.
- There are a number of diagnostic hypotheses that might explain the principal complaint.
- Some of the diagnostic hypotheses can be ruled out because they are: (1) unable to explain the principal complaint in this instance, or (2) implausible independent of what they might explain.
- The diagnostic conclusion is the best of the plausible hypotheses that are capable of explaining the principal complaint.

This argument form is the logical structure of the diagnostic task. It can be thought of as a means of justifying diagnoses. As such, it suggests specific ways a diagnostic conclusion might be wrong.

Suppose the diagnostic conclusion turns out to be wrong. What happened to the true answer? That is, why did the true, or correct, answer *not* turn out to be the best explanation? Based on the logical structure of diagnosis, given above, the diagnostic conclusion can only be wrong for one or more of the following reasons:

1. There is something wrong with the principal complaint. Either it is (1) not really present or does not need to be explained, or (2) incomplete, there are other things that should be explained by the diagnostic conclusion.
2. The true answer was not on the list of diagnostic hypotheses thought to have the potential of explaining the principal complaint.
3. There is an error in ruling out.
   (a) The true answer was ruled out. It was mistakenly thought (1) to be implausible or (2) not to explain the data.

(b) The wrong answer (the one given) was not ruled out. It was mistakenly thought (1) to be plausible or (2) to explain the data.

4. There is an error in choosing the best of the plausible explanations. Either (1) the wrong answer appears to be better than it is, or (2) the true answer appears to be worse than it is.

The source of these errors might be found in either missing or faulty knowledge as well as in various problems with the data itself.

Many users' questions can be interpreted as attempts to ensure that the conclusion is correct. Thus, corresponding to each source of potential error there is a class of questions, each seeking reassurance that a particular kind of error was not made. This analysis tells us that if we build a knowledge-based system and claim it does diagnosis, we can expect it to be asked the following kinds of questions.

1. Is the principal complaint really present or abnormal?
2. Does the principal complaint contain all the important data?
3. Was a broad enough set of explanatory hypotheses considered?
4. Has some hypothesis been incorrectly ruled out?
5. Could some hypothesis explain a finding that the system thought could not?
6. Was some hypothesis not ruled out that should have been?
7. Is it possible that the hypotheses in the diagnostic conclusion do not really explain the findings?
8. Might the hypotheses in the diagnostic conclusion be rated too high?
9. Has some hypothesis been underrated?

Furthermore, these questions express the only reasonable concerns that arise *solely because* it is a diagnosis system. We are not suggesting that all questions will be in exactly one of these classes, some may refer to many of these concerns, others are not specifically about diagnosis.

4.2 Using the Logical Structure for Explanation

Any diagnostic system will have some means of achieving the diagnostic goals specified in the logical structure given above. Otherwise it will fail, in some respect, to be a diagnostic system. The diagnostic question classes are derived from the diagnostic goals, so the first step in building an explanation component is to map the diagnostic question classes onto the program. That is, each question class (say, "Is it possible that the hypotheses in the diagnostic conclusion do not really explain the findings?") is mapped onto the the part of the system responsible for achieving the corresponding goal (in the example, the part that determines the symptoms a diagnostic hypothesis explains). This way the questions can be answered by the part of the system that made the relevant decisions to explain how the decision helps achieve the goal. In order for this to work, the explainer needs a way of mapping users' questions into the appropriate question classes.

User: What antibody in the conclusion explains the following test result:

(164 Coombs 3+)

Red: The finding:

(164 Coombs 3+)

is explained by:

antiS

Fig. 10. An Explanation From RED

Let us briefly consider an example from a diagnostic system called Red. In order to give blood to patients who need it, a hospital blood bank must select compatible donor blood. A part of this decision involves finding out what antibodies a patient has. Red is a system that aids in this antibody-identification problem. This is a kind of diagnostic problem since the data is a set of test results to be explained and the antibodies are used to explain them. One type of question that people ask of Red is what antibody in Red's conclusion explains a particular test result. This question is an instance of the question class defined by: "Is it possible that the hypotheses in the diagnostic conclusion do not really explain the findings?" This is derived from the potential error that the answer given does not actually explain the data. This, in turn, is derived from the diagnostic goal of explaining the data. So the question ("What explains a particular test result?") is directed to the component of Red that chooses antibodies to explain particular elements of data. It produces an explanation such as the one in Fig. 10. *The "(164 Coombs 3+)" is the notation for a test result and "antiS" is shorthand* for "antibody to the S antigen". This process of mapping the question to the part of the system that can answer it is not done automatically in Red. The logical structure of diagnosis was used in building Red's explanation component, but the mapping is hard-coded in the program. Tanner [36] describes explanation for Red in more detail while Red itself was fully reported by Josephson, et al. [20]

The logical structure of diagnosis presented here is a common view of the diagnostic task [12, 17, 20, 28, 29, 30]. Not all approaches to diagnosis will share this view. In fact, there is one common competing view - diagnosis as description, i.e., the goal of diagnosis is to describe the patient's state, not to find a cause for the symptoms. But if users and systems agree to a logical structure, it can be used to develop explanation for diagnosis in the manner we describe. The details will change if the model changes, but the method, and the idea, of using the logical structure to develop explanations remains.

5 Knowledge Justification: Relating Problem Solving to Causal Models

The integration and use of causal models in compiled problem-solving systems has become increasingly prevalent. Xplain was probably the first system to pro-

vide explanations of problem-solving knowledge by showing how it was obtained by compilation from other knowledge about the domain [26, 35]. Our work on functional representations (FR) [31] is similar in showing how to compile diagnostic programs from functional representations of mechanical devices. Following on this, Keuneke's work [23] showed how to use FR for justifying diagnostic conclusions, which we describe in this section.

5.1 Background

Methods that carry out problem-solving tasks need knowledge of certain kinds and in particular forms. For example, establish-refine, a method for hierarchical classification, requires knowledge relating descriptions of situations to descriptions of classes (see Sect. 2). If knowledge is not available in this form, it must be derived from some other knowledge. We refer to this derivation process as compilation [31, 35, 15], and to knowledge in the desired form as compiled knowledge [7]. The "other knowledge" has sometimes been called deep knowledge, but it is not necessarily deeper or better, only less compiled relative to the method that needs it. The compiled knowledge can be justified by referring to the knowledge from which it was compiled.

As with any compiled knowledge, compiled diagnostic knowledge can be justified by referring to the compilation process. Diagnosis also admits an interesting variation on this type of justification. If a system for diagnosing faults in a mechanical device is compiled from a causal model of the device, then its diagnostic conclusions can be related to observations using the causal model. This justifies the conclusion and validates the compiled knowledge that produced it. The causal model could be used to perform diagnosis, and systems have been built that do this [11, 27, 38], but for complex devices the large amount of causal information makes the diagnostic task very difficult. In most diagnostic systems, the causal knowledge is compiled for greater expertise and optimum diagnostic performance. Then, if a causal story can be put together, using the hypothesized diagnostic answer as a focus, we get the advantages of both worlds: the computational benefits of compiled knowledge to obtain the diagnostic answer, as well as the causal model to validate it.

In a diagnostic context, given the symptoms and the diagnostic conclusion, Keuneke showed how to use a causal model to justify the diagnosis at various levels of detail. In many situations a similar method will work to justify individual rules in the knowledge base. Wick [39] developed a related idea: justifying a conclusion in terms of the standard arguments used by domain experts. Both Wick and our work using FR produce justifications by reference to knowledge not used, perhaps not even needed, to produce the solution. However, one important difference is that justifications come from knowledge that, in principle, could be used to compile diagnostic problem solvers while Wick is not committed to any particular relationship between justification knowledge and problem-solving knowledge. The intent of Keuneke's [23] research was to continue efforts in the development of a device- and domain-independent representation capable of

modeling the core aspects of device understanding; the extended goal is a cognitive model of device understanding. Although this work was driven by the task of explanation, the representation was designed to provide the basic primitives and organization for a variety of problem-solving tasks.

The Functional Representation. Initial efforts to generate causal justifications [5, 6, 18, 24] focused on enhancing Sembugamoorthy and Chandrasekaran's FR [31] to provide a representation with the necessary organization and primitives[7].

Functional Representation is a representational scheme for the functions and expected behavior of a device. FR represents knowledge about devices in terms of the functions that the entire device is supposed to achieve and also of the sequence of causal interactions among components that lead to achievement of the functions. FR takes a top-down approach to representing a device, in contrast to the bottom-up approach of behavior-oriented knowledge representation and reasoning schemes [10, 14]. In FR, the function of the overall device is described first and the behavior of each component (its causal process) is described in terms of how it contributes to the function[8]. Figures 11 and 12 illustrate the top-level representation of a chemical processing plant.

In this representation, a device's *function* is its intended purpose. Functions describe a device's goals at the device level. For example, the function of a chemical processing plant is to produce a certain product. It has components for supplying the reactants, stirring the substance, extracting the product, and so forth. But generating the product is a function of the device as a whole.

Functions are achieved by *behaviors* or *causal processes*. In the chemical processing plant example, the substance is produced by the causal sequence of (1) input of the reactants into the reaction vessel, (2) allowing reactant contact, and then (3) extracting the product from the reaction vessel. In short, the functions are *what* is expected; the behaviors are *how* this expected result is attained. In FR, behaviors are represented as causal sequences of transitions between partial states or predicates (e.g., `(present reactants rxvessel)`).

The device is represented at various levels. The topmost level describes the functioning of the device by identifying which components and more detailed causal processes are responsible for bringing about the various state transitions. If a transition is achieved using a function of a component, the next level describes the functioning of this component in terms of the roles of its subcomponents, and so on. Ultimately, either by tracing through more detailed causal processes or by expanding the causal processes of functional components, all the functions of a device can be related to its structure and the function of the components within this structure.

Enhancing the Functional Representation. Early explanation work [5] simply used the FR as a tool to answer questions such as: (1) Why is this device

[7] For a more current formal treatment of the representation, see [19]

[8] A function's causal process is represented in the machine by its causal process description (CPD).

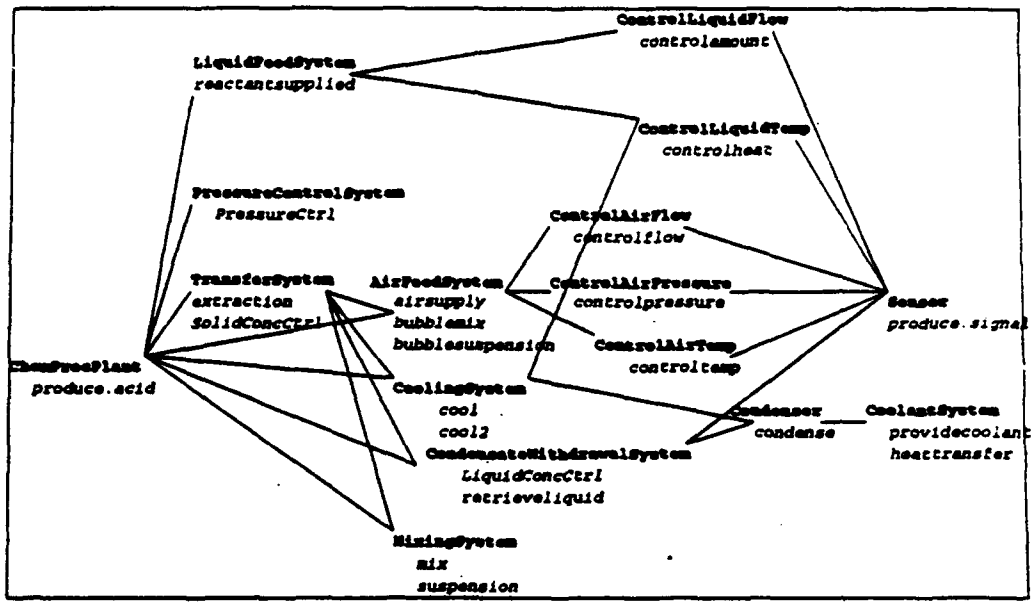
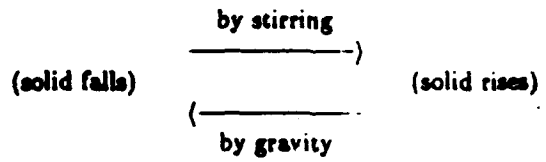


Fig. 11. Functional Component Hierarchy

needed? (2) What subcomponents does this device require? (3) What does this *function accomplish? (4) Why or where is this function used in the device?* (5) How is this function achieved?

Later, enhancements to the FR allowed the representation of state and behavior abstractions [25, 23]. Abstract schema specifications, and the ability to make transitions between abstraction levels, is useful for providing different levels of explanation.

For example, suppose there exists a solid/liquid mixture in which action is being taken to keep the solid from the bottom of the container. One might witness the following causal loop:



Here an observer could follow the loop any number of times, but somewhere one takes a conceptual jump and identifies the dynamic process (solid fall, solid rise, solid fall, solid rise...) as a *state* at a different behavioral level, i.e., identification of the process state (solid suspended). In doing so, one is identifying a new phenomenon; the observer is packaging a process and seeing it from a higher

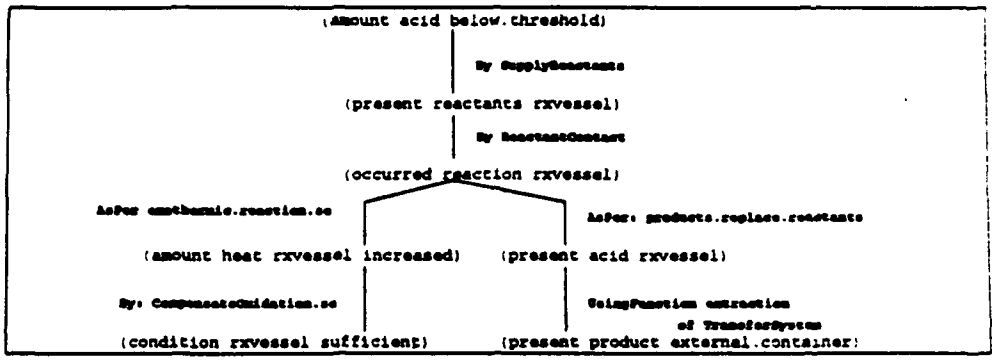


Fig. 12. CPD *oxidation* for Function *produce.acid* of a Chemical Processing Plant

conceptual viewpoint. These types of conceptual transitions are commonplace in one's understanding of a device's behavior - especially in cyclic or repeated behaviors. Nevertheless, past methods of behavior abstraction (detail suppression) did not explicitly address the representation of such phenomenon.

For researchers interested in building models of devices solely to predict behavior at a given level of detail, these abstractions will not be helpful. Instead, these abstractions provide the ability to tell a higher level story. Prediction is not driven solely by constraints of structure and low-level processes, but can be enriched and focused by knowledge of abstract processes and the inferences they dictate.

Additional enhancements include establishing a taxonomy of function types. Each function type indicates different procedures for simulation, different functional capabilities, different expectations, and thus different knowledge specifications for representation and explanation. Function types include:

1. **ToMake:** achieves a specific partial state
2. **ToMaintain:** achieves and sustains a desired state
3. **ToPrevent:** keeps a system out of an undesirable state
4. **ToControl:** gives a system power to regulate changes of state via a known relationship.

More details on the knowledge distinguishing each type, explicit specifications of the function types, and the information processing distinctions each type makes is provided in [23, 22].

The structure of the functional representation, organized around functional packages, provides focus through which simulation and the identification of structural cause can be determined (i.e., given changes in function, what changes in structure could be hypothesized to account for them?). Since, at some level, most problem-solving tasks dealing with devices are concerned with either the achievement of function, or consequences of the failure to achieve function, a functional description and reasoning power proves useful. The use of the representation in

diagnosis seems especially appropriate since diagnosis centers around determining the change in structure that resulted in some malfunction.

6 Causal Explanation: The Problem Statement

To illustrate the use of the representation, we pose the following problem: Given a set of observations and a diagnostic hypothesis, attempt to construct an explanation in the form of a causal story which starts with a diagnostic hypothesis and ends with one or more of the observations to be explained. In the following, we examine how a functional representation can be used for this purpose. Technical definitions of a few terms may be useful:

Observations: observable states, including a subset which are malfunctions of the device subsystems or components. The following distinctions about observations are useful:

- **Symptoms:** abnormal states which are indicative of malfunctions and trigger the diagnostic process, e.g., specification of a drop from normal pressure.
- **Malfunctions:** observations which *are* malfunctions, e.g., specification of a faulty pressure regulator. Malfunction observations are generally also symptoms.
- Observable states which provide information about the device but do not directly correspond to abnormalities, e.g., specification of temperature or pressure readings. Typically in a complex system, a large number of observations are used in the diagnostic process which provide focus for the problem-solving but do not necessarily indicate problems (e.g., sensor readings).

Diagnostic Hypotheses: the set of malfunctioning components or missing (but expected) relationships between components. Each in the latter should sooner or later, manifest itself as the malfunction of a subsystem within which the components lie.

Causal Explanation: Normally one expects a diagnosis to causally "explain" the symptoms, even though in general the diagnosis actually should explain all the observations. The explanation provided here takes any given set of observations to be explained and tries to propose a causal path from the diagnostic hypothesis to these observations.

The explanation sought can be formally stated as follows:

$$\text{diagnostic hypothesis} \rightarrow x_1 \ldots \rightarrow x_i \ldots \rightarrow x_N$$

where each x_i is either (1) an internal state which is causally relevant in producing an observation, but is itself not a malfunction, (2) a component or subsystem malfunction, or (3) an observation at the device-level. The explanation system developed in this work produces explanation chains where the members are limited to the last two, i.e., malfunctions or observations, *unless* the causally relevant internal state has been provided explicitly as a state that needs to be explained, i.e., as input to the casual explanation system.

6.1 Generating the Malfunction Causal Chain

In the same way a functional representation provides an organization to allow simulation of how *expected* functionalities are achieved, it can also serve as a backbone to trace the effects of not achieving certain functions - thus identifying potential malfunctions.

The organization of a functional representation gives both forward and backward reasoning capability, i.e., it can trace from the hypothesized malfunction to the observed malfunctions and symptoms (forward), or it can trace from observed malfunctions to the hypothesized malfunction (backward). Because both the observations and the diagnostic hypotheses have been identified once diagnosis is complete, the functional representation could potentially be used to perform either form of control. This section provides an algorithm which demonstrates the forward simulation potential[9].

Specifically, if device A is malfunctioning, then devices which use device A (say devices B and C) have a high probability of malfunctioning as well. Similarly, devices which use B and C may malfunction, etc. The malfunction causal chain is achieved through the following algorithm which has been condensed to illustrate main points.

1. - Set Observations to the symptoms and malfunctions to be explained,
   - Set MalfunctionList to the hypothesized malfunction set provided by the diagnosis,
   - Initialize MalfunctionObject to an individual malfunction in this set (diagnosed hypotheses and their relationship to observations are considered individually)
2. Find all functions which made use of the function which is malfunctioning (MalfunctionObject), call this set PossibleMalfunctions,
3. For each element in PossibleMalfunctions (call the specific function PossMal) consider the significance of the effect of MalfunctionObject on the function.
   - if no effect on PossMal then remove from PossibleMalfunctions - MalfunctionObject is not causing future problems. Consider the next element in PossibleMalfunctions.
   - *else maintain* (Malfunction — Malfunction) explanation chain; MalfunctionObject is now known to cause a malfunction to PossMal. Specifically MalfunctionObject — PossMal is appended to chain. Note that this step will ultimately place any potential malfunctions in a malfunction chain, including those which are in the set of Observations. Continue.
4. Check the states in the causal process description of the affected PossibleMalfunction. Would noncompletion of these states explain any symptom(s) in Observations?

[9] Note that since the explanation generation mechanism uses expected functionalities and their causal processes rather than all behaviors that could possibly be generated, the problem space is bound and thus focused.

- if yes, append to ExplainedSymptoms and print the chain which led to this symptom. Complete the malfunction explanation chain by continuing.

5. Set MalfunctionObject to PossMal. (MalfunctionObject $\Leftarrow$ PossMal)
6. Repeat process from step 2 until all symptoms are in ExplainedSymptoms or the top level causal process description of the device has been reached.
7. The Process from step 1 is repeated until all elements of MalfunctionList have been considered.

Step 2 is easily accomplished through the component hierarchy of the functional representation (example in Sect. 6.2). Step 3 and 4 are more intricate and involve knowledge of function type and the achievement of the intended causal processes.

For example, in step 3, to determine the effects of a malfunction on other functions, one must consider the possible consequences of malfunctioning components. In general, the malfunction of a component in a device can cause one or more of the following three consequences:

NOT Function: expected results of the function will not be present. Given that the malfunction is not producing the expected results within the causal process, what states in those causal processes will not occur, and will lack of this functionality cause the malfunctions of functions in which the malfunctioning component was used?

Parameter Out-of-Range: expected results of the function are affected, but behavior is still accomplished to a limited degree. Sometimes components may be considered malfunctioning yet can still perform the behavior (or value of some substance parameter) to the extent needed for future use.

New Behaviors: the malfunction results in behaviors and states which were not those intended for normal functioning.

The determination of whether a proposed malfunction can explain a symptom, step 4 in the explanation algorithm, can be established by a number of means:

1. Check each state in the causal process description where the malfunctioning component is used to see if there is a direct match between a symptom and *not* achieving an expected state.
2. Check to see if the function which is malfunctioning has an explicit malfunction causal process description and if the symptom is included therein. [10]
3. Check to see if side effects of the functions causal process description refer to the symptoms.
4. Check each state in the malfunction causal process description and its provided clause to see if expected states point to general concepts or generic

[10] See [23] for knowledge of function type and detail on functions with explicit malfunction causal processes.

classes of behavior (such as leak, flow, continuity) and if the symptom pertains to or is explained by such concepts.

5. If the malfunction is a malformation, i.e., the malfunction is described as a malformation of a particular physical component, perform deep reasoning (e.g., qualitative physics) to see if malformation could cause the symptom.

The first three are implemented and currently used for the explanation generation; the means to perform the last two are research in progress.

6.2 Representation of a Chemical Processing Plant

This section provides the output for an example explanation in the domain of Chemical Processing Plants. Reference to Fig. 11 (in Sect. 5.1) will assist the reader in following the causal explanation chains given by the algorithm. The hierarchy in Fig. 11 shows a partial representation of the functional components with their intended functions (functions are specified under component names). The top level function, *produce.acid*[11], is achieved by the causal process *oxidation* shown in Fig. 12. It should be noted that the function hierarchy is *generated* given the causal processes used to achieve functions of the functional component. For example, the Chemical Processing Plant uses the functional components LiquidFeedSystem, AirFeedSystem, TransferSystem, etc. in the process *oxidation* which represents the causal chain used to achieve the function *produce.acid*; the TransferSystem uses the functional components AirFeedSystem, MixingSystem, etc. in its causal process to achieve the function *extraction*, and so on.

The Problem. The Coolant System (identified at the right of Fig. 11) is used to provide coolant water to a Condenser so that it can be used to transfer heat from the vapor in the Condenser (see Fig. 13). Suppose the coolant water has been completely cut off. A diagnostic system has concluded that a malfunction of the function *provide.coolant* of the Coolant System explains the symptoms of NOT (present product external.container) and NOT (temperature rxvessel at.threshold). Specifically, MalfunctionObject is {*provide.coolant* of Coolant System} and the Observations to be explained are {NOT (present product external.container), NOT (temperature rxvessel at.threshold) }. The system produces the following three casual stories.

Causal Story 1: Generation of Causal Connections. The causal process SupplyReactants uses the functions *retrieveliquid* and *LiquidConcCtrl*, in addition to the LiquidFeedSystem and AirFeedSystem. The explanation system generates the following:

```
The symptom
NOT (present product external.container)
is explained by the following chain:
```

[11] The acid produced is a solid, terephthalic acid.

```
  NOT provide.coolant causes
  malfunction in condense causing
  malfunction in retrieveliquid causing
  malfunction in LiquidConcCtrl causing
  problems in behavior SupplyReactants
  which is used in behavior oxidation and
  indicates malfunction of the top level
  function and results in
  NOT (present product external.container)
---
  The following symptoms are not explained:
  NOT (temperature rxvessel at.threshold)
```

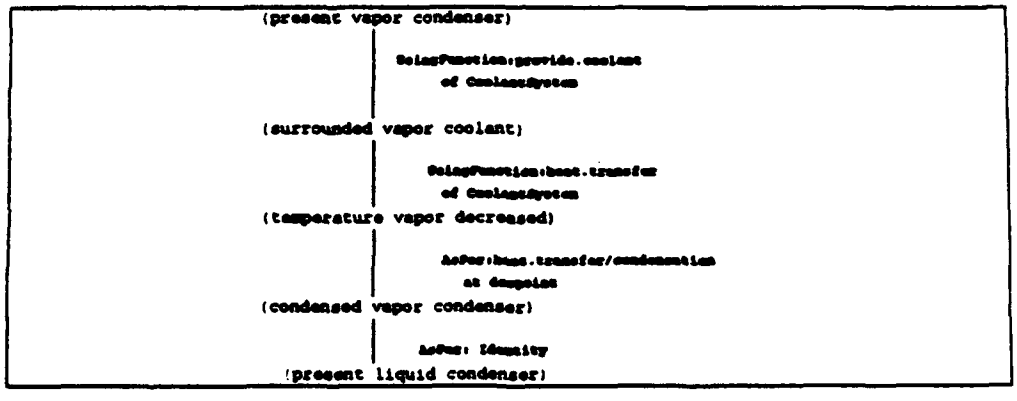


Fig. 13. CPD: *RemoveHeat* of Function *Condense*

The idea here is that if the required amount of reactants is not available, the product is not produced as desired and thus can not be retrieved. The explanation system generates this chain by using the following information: *Provide.coolant* caused a malfunction in *condense* because it caused a failure in *condense*'s behavior. A malfunction in *condense* caused a malfunction in *retrieveliquid* because its achievement was required to attain the desired behavior for *retrieveliquid*. *Retrieveliquid* caused a malfunction in *LiquidConcCtrl* because it was needed to provide the preconditions for *LiquidConcCtrl* and it preceded the use of *LiquidConcCtrl* in the behavior SupplyReactants. SupplyReactants was used in the causal process Oxidation, Fig. 12, to achieve the state (present reactants rxvessel). This state was necessary for the completion of the behavior and thus non-achievement here denotes non-achievement of further states in the behavior, particularly NOT (present product external.container).

Causal Story 2: The Use of Side Effect Inspection. The explanation system continues and finds a causal connection for the second symptom, NOT (temperature rxvessel at.threshold).

```
The symptom
NOT (temperature rxvessel at.threshold)
is explained by the following chain:
 NOT provide.coolant causes malfunction
 in condense causing problems in behavior
 removeheat of function cool
```

Since *cool* is not a top level function of the chemical processing plant, the trace continues until all consequences are determined.

```
The symptom
NOT (temperature rxvessel at.threshold)
is explained by the following chain:
 NOT provide.coolant causes
 malfunction in condense causing
 malfunction in cool causing problems
 in behavior compensate.oxidation.se
 a notable side effect behavior used in
 oxidation and indicates
 NOT (temperature rxvessel at.threshold)
---
 The following symptoms are not explained
 (present product external.container)
```

Notice that this explanation identifies that the symptom was observed in a *side effect* behavior (compensation for effects of the reaction) rather than a behavior of the main functionality (production of acid).

The statement of which symptoms are not explained indicates those that were not explained in the specific causal chain. A final statement is made when the system has inspected all pertinent causal chains (as seen in the next causal story).

Causal Story 3: Using Subfunction Connections for Causal Focus. The final causal path is achieved via causal connections obtained specifically through the knowledge of subfunctions. In its specification, the function *extraction* has a provided clause which specifies that the solid acid slurry must have the proper consistency so that flow through the extraction tube is possible. The function *SolidConcCtrl* is present in this device for the sole purpose of producing these conditions for *extraction*.

The purpose of *SolidConcCtrl* is to keep the solid suspended and the concentration in the reaction vessel at the proper consistency. In the Condensate-WithdrawalSystem, the *retrieveliquid* function uses the Condenser to retrieve

the condensate from the vapor produced. The *MixtureLevelCtrl* function then uses a feedback controller to maintain the flow and thus the desired amount of liquid in the reaction vessel – which ensures that the acid slurry has the proper consistency. If the liquid is not retrievable, then obviously the condensate flow cannot be controlled and consistency of the acid in the vessel is not maintained. The explanation system provides this explanatory story as follows:

```
One function affected by provide.coolant
is SolidConcCtrl which is a necessary
subfunction of extraction

 The symptom
 NOT (present product external.container)
 is explained by the following chain:
  NOT provide.coolant causes
  malfunction in condense causing
  malfunction in retrieveliquid causing
  malfunction in MixtureLevelCtrl causing
  malfunction in SolidConcCtrl causing
  malfunction in extraction causing
  malfunction in produce.acid causing
  NOT (present product external.container)
---
  All symptoms have been explained.
```

6.3 Discussion

The intrinsic limitations of a functional representation for explanation arise from its intrinsic limitations for simulation. The representation uses prepackaged causal process descriptions which are organized around the expected functions of a device. Simulations of malfunctioning devices are thus limited to statements of what expectations are "not" occurring.

This limitation effects the capabilities for explanation in two significant ways. First, the functional representation is not capable of generating causal stories of malfunctions which interact unless the device representation has this interaction explicitly represented. Similar problems regarding the interactions of malfunctions arise in diagnosis [33]. Secondly, "new" behaviors, i.e., behaviors which are not those intended for normal functioning but which arise due to a change in device structure, could potentially lead to symptoms which cannot be explained using the functional representation. Current research efforts focus on how a functional organization might be used to determine these new behavioral sequences, in addition to how conventional methods of qualitative reasoning may be integrated.

Additional Applications of a Functional Model

The idea of considering how devices work is a generally useful concept which provides a focus for reasoning about objects. Since goals can be attributed to many types of objects, a general representational language, focused around functionality, can potentially model an understanding of a variety of object types, i.e., truly a "device-independent" representation. In addition, the organization around functions helps to focus a reasoner's attention toward expected goals; something works like it does because it is meant to achieve a specific purpose. The practical uses of having a functionally oriented understanding of how something works can be seen in the following applications:

diagnosis: How something works provides information about what functions to expect from working models, and thus implicitly knowledge of malfunctioning models. This helps to enumerate malfunction modes and to derive what observable consequences follow for a given malfunction.

learning: In diagnosis, if a hypothesis has been made and a causal chain cannot be found that connects the hypothesis to the symptoms, a learning process could be triggered. Specifically, a diagnosis which cannot be causally connected to the symptoms might cause suspicion, not only about the diagnostic result, but also about the knowledge used in the diagnostic process. Use of the malfunction causal explanation capabilities can help explicate erroneous malfunction hypotheses and aid in pointing to alternatives. [37]

repair/replacement: Knowledge of how a device works indicates knowledge of its teleology. Replacement with objects of like teleology can be considered.

design/redesign: Knowledge of what functionalities are desired can point the designer to necessary components. [16, 19]

planning: The representation of plans (as devices) provides an understanding of how the plan's goals are achieved. [5]

determination of optimum use: Knowledge of how a device works can provide information regarding how to use the device to its maximum potential.

analogy: Organizing knowledge of how one object works provides links for determining how a similar object might operate.

prediction: Knowledge of expected functionalities focuses reasoning for determining what will happen in a device. [32]

simulation: Simulation of expected device behavior is useful for problem solving, in particular, design. [34, 19]

explanation: Having the knowledge of how something works allows one to simulate and explain the mechanism, i.e., tutorial purposes.

8 Conclusion

In this paper we have surveyed the work done at the Ohio State Laboratory for AI Research on knowledge systems explanation. We consider the explanation problem to have three aspects: the explanation content, the form of presentation, and the manner of presentation. We have concentrated on the explanation

content, which we see as having four parts: explaining problem-solving steps, strategy, and task, and justifying knowledge. Most of our work on these has been guided by GT theory – any task can be accomplished by many different methods, the combination of a particularly appropriate, domain-independent, method with a task is called a generic task. GT research has identified several generic tasks and a knowledge system that uses a generic task can explain its steps and its strategy, since strategy is an aspect of the method. By combining generic tasks with a theory of tasks, independent of method, it is possible to give explanations that show how a system's method achieves the task goals. Using the functional representation, also developed at the LAIR, to represent general purpose knowledge in the knowledge system's domain we can justify its problem-solving knowledge by showing how it was derived. Individually, each of the efforts described here solves a few problems and leaves many issues unaddressed. Taken as a whole, they represent an attempt to explore the wide range of roles that knowledge plays in explanation – knowledge about tasks, methods and strategies, system design, background domain knowledge, and memory for particular problem-solving instances – and the benefits of explicitly representing these kinds of knowledge.

9 Acknowledgments

This work has been supported by the Air Force Office of Scientific Research (grant 89-0250), the Defense Advanced Research Projects Agency (contracts F30602-85-C-0010 and F49620-89-C-0110), and the National Heart Lung and Blood Institute (grant HL-38776). In addition we would like to thank John Josephson who led the MPA project and provided insightful comment on the rest of the work reported here; the other members of the MPA team: Dean Allemang, Matt DeJongh, Ron Hartung, and Dave Herman; and our friends and colleagues at the LAIR who have contributed to the ideas presented here through their work, discussions, and friendship. We also thank Bill Swartout and Cecile Paris for their helpful comments on an earlier draft. These individuals do not necessarily endorse the entire contents of this paper, however. The authors accept full responsibility for that, including any inadvertent errors.

References

1. Brown, D. C., Chandrasekaran, B.: Design Problem Solving: Knowledge Structures and Control Strategies. Morgan Kaufmann, Inc., San Mateo, CA, 1989
2. Bylander, T., Mittal, S.: CSRL: A language for classificatory problem solving and uncertainty handling. AI Magazine, 7(3):66–77, August 1986
3. Chandrasekaran, B.: Design problem solving: a task analysis. AI Magazine, 11(4):59–71, Winter 1990
4. Chandrasekaran, B.: Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. IEEE Expert, 1(3):23–30, Fall 1986

5. Chandrasekaran, B., Josephson, J., Keuneke, A.: Functional representation as a basis for explanation generation. Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, pages 726–731, 1986
6. Chandrasekaran, B., Josephson, J. R., Keuneke, A. M., Herman, D.: Building routine planning systems and explaining their behavior. International Journal of Man-Machine Studies, 30:377–398, 1989
7. Chandrasekaran, B., Mittal, S.: On deep versus compiled approaches to diagnostic problem solving. International Journal of Man Machine Studies, 19:425–436, 1983
8. Chandrasekaran, B., Tanner, M. C., Josephson, J. R.: Explaining control strategies in problem solving. IEEE Expert, 4(1):9–24, Spring 1989
9. Clancey, W. J.: Heuristic classification. Artificial Intelligence, 27(3):289–350, December 1985
10. Crawford, J., Farquhar, A., Kuipers, B.: QPC: a compiler from physical models into qualitative differential equations. Proceedings of the 8th National Conference on Artificial Intelligence, pages 365–372, 1990
11. Davis, R., Shrobe, H., Hamscher, W., Wieckert, K., Shirley, M., Polit, S.: Diagnosis based on description of structure and function. Proceedings of the 2nd National Conference on Artificial Intelligence, pages 137–142, Pittsburgh, PA, 1982
12. deKleer, J., Williams, B. C.: Diagnosing multiple faults. Artificial Intelligence, 32(1):97–130, April 1987
13. Engelman, C., Millen, J. K., Scarl, E. A.: Knobs: An Integrated AI Interactive Planning Architecture. Technical Report DSR-83-162, The MITRE Corporation, Bedford, MA, 1983
14. Falkenhainer, B., Forbus, K.: Setting up large-scale qualitative models. Proceedings of the 7th National Conference on Artificial Intelligence, pages 301–306, 1988
15. Goel, A.: Knowledge compilation: a symposium. IEEE Expert, 6(2):71–73, April 1991
16. Goel, A., Chandrasekaran, B.: Functional representation of designs and redesign problem solving. Proceedings of the 11th International Joint Conference on Artificial Intelligence, 1989
17. Harvey, A. M., Bordley, J. III: Differential Diagnosis, the Interpretation of Clinical Evidence. W. B. Saunders, Philadelphia, 1972
18. Herman, D., Keuneke, A., Tanner, M. C., Hartung, R., Josephson, J.: MPA: A mission planning assistant in the Knobs domain. Expert Systems: Proceedings of a Workshop, pages 103–116, Pacific Grove, CA, April 16–18 1986
19. Iwasaki, Y., Chandrasekaran, B.: Design verification through function- and behavior-oriented representations: bridging the gap between function and behavior. Proceedings of the Conference on Artificial Intelligence in Design, 1992
20. Josephson, J. R., Chandrasekaran, B., Smith, J. W. Jr., Tanner, M. C.: A mechanism for forming composite explanatory hypotheses. IEEE Transactions on Systems, Man, and Cybernetics, SMC-17(3):445–454, May/June 1987
21. Josephson, J. R., Smetters, D., Fox, R., Oblinger, D., Welch, A., Northrup, G.: Integrated Generic Task Toolset: Fafner Release 1.0, Introduction and User's guide. Technical Report 89-JJ-FAFNER, Lab. for AI Research, Ohio State Univ., Columbus, OH, June 1 1989
22. Keuneke, A.: Device representation: the significance of functional knowledge. IEEE Expert, 6(2):22–25, April 1991
23. Keuneke, A.: Machine Understanding of Devices: Causal Explanation of Diagnostic Conclusions. PhD thesis, The Ohio State University, Columbus, Ohio, 1989

24. Keuneke, A., Allemang, D.: Exploring the "No-Function-In-Structure" Principle. Journal of Experimental and Theoretical Artificial Intelligence, 1:79–89, 1989
25. Keuneke, A., Allemang, D.: Understanding Devices: Representing Dynamic States. Technical Report, The Ohio State University, 1988
26. Neches, R., Swartout, W. R., Moore, J. D.: Enhanced maintenance and explanation of expert systems through explicit models of their development. IEEE Transactions on Software Engineering, SE-11(11):1337–1351, November 1985
27. Patil, R. S.: Causal Representation of Patient Illness for Electrolyte and Acid-Base Diagnosis. PhD thesis, MIT Lab for Computer Science, TR-267, Cambridge, Massachusetts, 1981
28. Pople, H. E.: The formation of composite hypotheses in diagnosic problem solving. Proceedings of the 5th International Joint Conference on Artificial Intelligence, pages 1030–1037, Cambridge, MA, 1977
29. Reggia, J.: Diagnostic expert systems based on a set covering model. International Journal of Man-Machine Studies, 19(5):437–460, November 1983
30. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence, 32(1):57–95, April 1987
31. Sembugamoorthy, V., Chandrasekaran, B.: Functional representation of devices and compilation of diagnostic problem solving systems. J. L. Kolodner and C. K. Riesbeck, editors, Experience, Memory, and Reasoning, pages 47–73, Lawrence Erlbaum Assoc., Hillsdale, New Jersey, 1986
32. Sticklen, J.: MDX2, An Integrated Medical Diagnostic System. PhD thesis, The Ohio State University, 1987
33. Sticklen, J., Chandrasekaran, B., Josephson, J.: Control issues in classificatory diagnosis. Proceedings of the 9th International Joint Conference on Artificial Intelligence, pages 300–306, IJCAI, 1985
34. Sun, J., Sticklen, J.: Steps toward tractable envisionment via a functional approach. Second AAAI Workshop on Model Based Reasoning, pages 50–56, 1990
35. Swartout, W. R.: Xplain: A system for creating and explaining expert consulting programs. Artificial Intelligence, 21(3):285–325, September 1983
36. Tanner, M. C.: Explaining Knowledge Systems: Justifying Diagnostic Conclusions. PhD thesis, Dept. of Computer and Information Science, Ohio State University, Columbus, OH, March 1989
37. Weintraub, M.: An Explanation Approach to Assigning Credit. PhD thesis, The Ohio State University, Columbus, Ohio, 1991
38. Weiss, S. M., Kulikowski, C. A., Amarel, S., Safir, A.: A model-based method for computer-aided medical decision-making. Artificial Intelligence, 11:145–172, 1978
39. Wick, M. R., Thompson, W. B., Slagle, J. R.: Knowledge-Based Explanation. TR 88-24, Computer Science Dept., Univ. of Minn., Minneapolis, MN, March 1988

This article was processed using the LaTeX macro package with LLNCS style

Functional Representation as Design Rationale

B. Chandrasekaran, Ohio State University
Ashok K. Goel, Georgia Institute of Technology
Yumi Iwasaki, Stanford University

The design process involves exploring design spaces, simulating and verifying candidate designs, and possibly redesigning and repeating the cycle. The body of information that explicitly records the design activity and the reasons for making choices (and reasons for *not* making some choices) is called the design rationale (DR). As more of the design process gains computational support, some designers are focusing on the tasks of defining the components of DR and casting it into a form that can be recorded and manipulated computationally.

Research is addressing what kinds of information DR should contain and how to express it. In a recent special DR issue of the journal *Human-Computer Interaction*,[1] MacLean et al.[2] proposed a semiformal notation called Questions-Options-Criteria (QOC) to represent the design space. As the space is explored, Questions identify key design issues, Options provide possible answers to these, and Criteria help assess the options. Lee and Lai[3] proposed a language called DRL, which provides a vocabulary for recording design alternatives, the evaluation space and criteria, and the argument structure in which design discussions are conducted.

Lee and Lai make a useful distinction between using DR as

(1) a record of the exploratory activity of the design team (along the lines of the information captured by the QOC formalism) and
(2) an account of how the designed artifact serves or satisfies expected functionalities.

The distinction is essentially one of describing the designer's activity (what alternatives were considered and what choices were made and why) versus describing the artifact's functioning. We consider the use of a representation called Functional Representation (FR) for describing how the device works (or is intended to work). Specifically, we wish to show how FR can be used to capture the causal component of DR. By that we mean the designer's (or the design team's) account of the causal interaction sequence between device components that leads to achieving device functions.

Some of the tasks for which DR can be used are

(1) *Controlling distributed design activity.* In concurrent engineering, the DR for design decisions made by one group can be used by other groups to avoid redundancy of effort and incompatible design choices.

(2) *Reassessing device functions.* During the period of device use, the compo-

Although a design rationale cannot be completely represented, Functional Representation is a good framework for describing causal components because it embodies a theory of how causal stories are understood.

0018-9162/93/0100-0048$03.00 © 1993 IEEE

nent values might drift or even change qualitatively. Users might examine DR to see whether the device can still be expected to achieve the desired functions. They might also examine it to evaluate deviations from the expected range of behavior.

(3) *Generating diagnostic knowledge.* DR can support the generation of diagnostic procedures, thus helping maintainability.

(4) *Simulating and verifying design.* DR can help evaluate whether the device will perform as intended. In particular, DR information can assist in focusing and controlling device simulation to verify expected functions.

(5) *Redesigning and case-based designing.* When it is desirable to change a device's function, much of the structure can be retained if the intended change is not drastic. DR can help identify the components, subsystems, or parameters needing change, thus avoiding a new design from scratch. Similarly, when a new device is being designed, previous designs can be examined for similar functionality. The design that requires the least modification can be chosen and modified (case-based design). DR can also be useful in identifying the design that is "closest" to the desired device and in noting where this design needs modification.

The FR language includes elements for capturing DR in a form that can be helpful in performing some of these tasks.

Functional Representation

As stated previously, FR is a representational scheme[4] for the causal processes that culminate in the achievement of device functions. (Not all devices are best viewed as achieving their functions by means of causal processes, which we discuss later.) FR takes a top-down approach to representing a device in the sense that the overall function is described first and the behavior of each component is described in the context of this function. This contrasts to the bottom-up approach taken in many behavior-oriented knowledge-representation and reasoning schemes. (See the sidebar.)

Figure 1 is a schematic of a device

Behavior-oriented approach to modeling devices

Various behavior-oriented schemes for modeling devices take an approach very different from that of Functional Representation (FR) to represent knowledge about behavior. The behavioral characteristics of each conceptually distinct physical phenomenon, including different aspects of component behavior or the physical phenomenon, are represented as an independent piece of knowledge. Such pieces are often called model fragments. Each fragment is a context-independent description of a physical phenomenon. This fragment typically consists of the conditions necessary for the phenomenon to take place, including the objects that must exist, the conditions they must satisfy, and the consequences. The latter include the functional relations that will hold among quantities, as well as other effects on quantities. Given a description of the physical structure of a device and the initial condition, a reasoning system selects the applicable model fragments and predicts the behavior of the entire device by composing the knowledge of the behavior of individual components and the physical processes through which they interact.

In contrast to the function-oriented approach taken by FR, the behavior-oriented approach makes that the behavior of each component or a physical process is represented in a context-independent manner in the sense that each description makes no implicit assumptions about the physical context in which the component is placed or about the function of the whole device of which it is a part. This view is articulated as the *no-function-in-structure principle* and *locality principle* by de Kleer and Brown.[3] The purpose of these principles in modeling devices is to ensure that a prediction of behavior generated in this manner will be objective in the sense that it will not be biased by any hidden assumptions about perceived functions of the whole device. These principles are also necessary if, confronted with the device under a new condition or an unfamiliar device consisting of components configured in a novel fashion, a system is to be able to reason about the behavior of the device by composing knowledge of individual components and their possible interactions.

A number of systems have been built based on one of the behavior-oriented approaches or on the function-oriented approach to modeling devices.[1,2] Though most of the model-based reasoning systems built to date are based on one of the behavior-oriented approaches described here, or on the function-oriented approach described in the main text, the two approaches are complementary, and much can be gained from combining them. Generating and understanding DR, especially if it concerns behavior, requires both types of knowledge: knowledge of intended functions and of the underlying structures and general physical principles. In engineering design, designers first conceptualize the function to be achieved by the artifact. Then, they formulate a conceptual causal mechanism to achieve the goal. Finally, they produce a particular design of the physical mechanism to implement the conceptual causal mechanism. The capability to simulate the behavior of the designed device solely on the basis of the design and knowledge of the general physical principles, regardless of the intended function, is extremely valuable in forecasting the consequences of design decisions. At the same time, knowledge of the device's intended function is indispensable in evaluating the design on the basis of such predictions.

References

1. J. de Kleer and J.S. Brown, "A Qualitative Physics Based on Confluences," *Artificial Intelligence*, Vol. 24, Nos. 1 to 3, 1984, pp. 7-83.
2. J. Crawford, A. Farquhar, and B. Kuipers, "QPC: A Compiler from Physical Models into Qualitative Differential Equations," in *Proc. Eighth Nat'l Conf. Artificial Intelligence*, AAAI Press/MIT Press, Cambridge, Mass., 1990, pp. 365-372.
3. B. Falkenhainer and K. Forbus, "Compositional Modeling: Finding the Right Model for the Job," *Artificial Intelligence*, Vol. 51, Nos. 1 to 3, 1991, pp. 95-143.

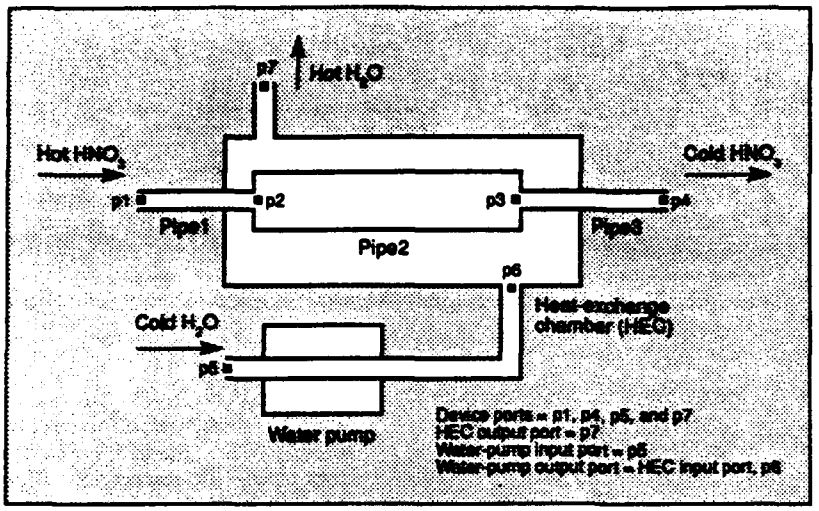


Figure 1. Schematic of a nitric acid cooler.

called a nitric acid cooler[5] (NAC). In FR, we represent the structure and function of a device and the causal processes that occur within it. We use as primitive notions the ideas of a system, its input, and its output behavior. A *device* is a system with some *intended* input-output relations, called *functions*. A *component* of a device is itself a system characterized by its functions.

Classes of functions and components can often be described by use of parameters. In the NAC example, component class "pipe(l, d)" describes pipes with length l and diameter d, while "pipe2" is a specific instance of "pipe(l, d)," with specific values for l and d. Similarly, the device NAC as a class has a function "cool-input-liquid(rate, temperature-drop)," where "rate and temperature-drop" are capacity parameters of the function "cool-input-liquid." A particular NAC can be identified by specific values for these parameters. Devices can have substances whose properties are transformed as part of their functions. Substances can be destroyed and new ones created. They can be physical (as in "nitric-acid") or abstract (as in "heat"). In the NAC example, the substance "nitric-acid" has the properties "temperature," "flow rate," and "amount of heat" (which itself is a substance).

Components are configured in specific structural relations to each other to compose a device. Components thus have ports, at which they join other components in certain relations. In an electrical circuit, components are "electrically-connected" at defined terminals. In the NAC example, the relations include "conduit-connection," "containment," etc. (The relational vocabulary can also include unintended relations, such as "electrical leakage between components." The components can be in unintended relations as a result of malfunctions.) The vocabulary of relations is domain-specific. The relation semantics are established by the domain laws that govern the behavior of the components in the given relations.

Device structure. A device's structure is a specification of the set of its components and their relationships. The components are represented by their names and by the names of their functions, which are all domain-specific strings. Variables can serve as component parameters. All components have ports to connect with other components. For example, the component type "pipe" might be written as "pipe(l, d; t_1, t_2)," where l and d are the length and diameter, and t_1 and t_2 are the input and output ports. The FR language uses

Figure 2. Keywords and syntax for device structure in Functional Representation.

```
STRUCTURE({DEVICE(<device-name>, <functional parameters>, <ports>)},
  COMPONENT(<component-name>, <component parameters>, <ports>),
  FUNCTION(<component>), RELATION(<relation[<component>, ... <component>]>)
```

```
STRUCTURE({DEVICE(NAC; cooling-capacity and temperature parameters, ports: p1, p4, p5, p7)}
COMPONENTS: pipe1(l1, d1; p1, p2), pipe2(l2, d2; p2, p3), pipe3(l3, d3; p3, Output)
    Heat-exchange-chamber(<dimensions>, Input-port, Output-port)
    Water-pump(Input, Output)
  FUNCTION(pipe1): Conduit (input, output)
  FUNCTION(Heat-exchange-chamber): exchange-heat(<parameters>)
  FUNCTION(Water-pump): ...
RELATIONS: COMPONENT(pipe2) contained-in COMPONENT(Heat-exchange-chamber)
  COMPONENT(pipe1) conduit-connected (pipe2) [PORTS: <information about ports>]
  COMPONENT(Water-pump) conduit-connected COMPONENT(Heat-exchange-chamber)
    [PORTS: <information about which ports of components are connected; e.g., Input-port of
    Heat-exchange chamber is the same as the Output of Water-pump>]
```

Figure 3. Structure of nitric acid cooler.

keywords for describing structure, as shown in Figure 2. The capitalized keywords are FR (and hence DR) terms.

Figure 3 describes the structure of NAC. The terms in italic are domain-specific names for functions, components, relations, etc. The FR interpreter treats them as strings. Additional domain-specific interpreters may be written that can use the italicized terms as meaningful keywords. For example, a mechanical *simulator* can use such terms as "contained-in" and "conduit-connected" to perform simulations. For the purpose of this exposition, they are to be understood in their informal, English-language meanings. The syntax of the Relations keyword is that an n-ary relation has n components. The Ports term indicates connected ports. Note that the components are described purely in terms of their functions. Components can thus, in principle, be replaced with structurally different but functionally identical components. Further, the components themselves can be represented as devices on their own terms.

States and partial states. A device state is represented as a set of state variables consisting of values of all variables of interest in the device description. State variables can be continuous or discrete. In particular, some of the variables can take the truth values $[T, F]$, that is, they are defined by predicates. An example of a continuous variable is "water temperature" in a device that uses water for cooling a substance. An example of a variable defined by a predicate is "open?(valve)" (not shown in the figures). This variable takes T or F as a value, depending on whether the valve is open or shut.

In describing functions and causal processes, we generally talk in terms of partial states of the device. These states are given by the values of a subset of state variables. For example, the partial state (call it "state1") of NAC (describing some relevant state variables at the input $p1$ of the device) can be given as

(substance: nitric acid; location (substance): $p1$,temperature (substance): $T1$).

"State2," describing the properties of nitric acid at location $p2$, differs only in the location parameter, while the partial state description, "state3," at location $p3$ is

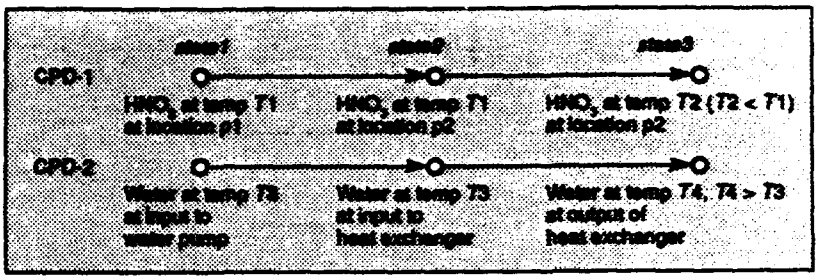


Figure 4. Causal process descriptions for a device nitric acid cooler (without link annotations).

(substance: nitric acid; location (substance): $p3$,temperature (substance): $T2$],

where $T2 < T1$. The language in which states are represented is not part of FR and is largely domain specific. In economics, the state variables would be entities such as "GNP" and "inflation-rate"; in nuclear plants, the entities would be "radiation-level," and so on. Goel[3] defined a state-description language useful for devices dealing with material substances that change locations, such as those in which substance flow is a useful notion. The state representation that we just used for NAC employs this language.

Causal process description. The intuitive idea here is that we understand how devices work by building a causal description of how they go through various states until the desired ones are reached. We explain the causal transitions between states by appealing to component functions and domain laws (such as scientific laws). The causal process description (CPD) is a directed graph with two distinguished nodes, N_{ini} and N_{fin}. Each node in the graph represents a partial description of a device state. N_{ini} corresponds to the partial state when the conditions for the function are initiated (such as when a switch is turned on). N_{fin} corresponds to the state when the function is achieved. Each link represents a causal connection between nodes. One or more qualifiers are attached to the links to indicate the conditions under which the transition will take place. One or more annotations can be attached to indicate the type of causal explanation to be given for the transition. The graph can be cyclic but must have a directed path from N_{ini} to N_{fin}.

Consider the NAC example again. Let nodes "state1," "state2," and "state3" correspond to the states of nitric acid at the input to "pipe1," at locations $p2$ and $p3$, respectively. Figure 4 depicts the CPD graph (without any annotations or qualifiers), describing what happens to the nitric acid and water as they flow through the chamber. We have represented the nodes in informal English, but they can be described more formally, similar to our previous description of "state1."

We have identified three types of annotation for explaining a causal transition: appealing to another causal process, to a function of a component, and to domain laws (so-called first principles). Let's elaborate on the three different types of knowledge used for explaining a causal transition.

(1) *By-CPD*. In explaining the transition "water heated → steam created," we can point to the causal process of "boiling," which is part of the commonsense knowledge of most humans. The agents for whom the explanation is intended will accept the explanation if they already understand the process or if the details of the process do not matter to their purposes. If the details do matter, they can ask for a more detailed explanation of the causal transitions involved in "boiling." In any case, this enables the process explanation to be hierarchically composed from other process explanations, shortening explanations at each level. Once "boiling" is understood, it can be reused, possibly after instantiating some parameters (such as the pressure at which boiling is performed and the liquid that is being boiled), whenever it is needed to explain other processes. Human domain expertise contains knowledge of a large number of such causal processes that can be parameterized and reused.

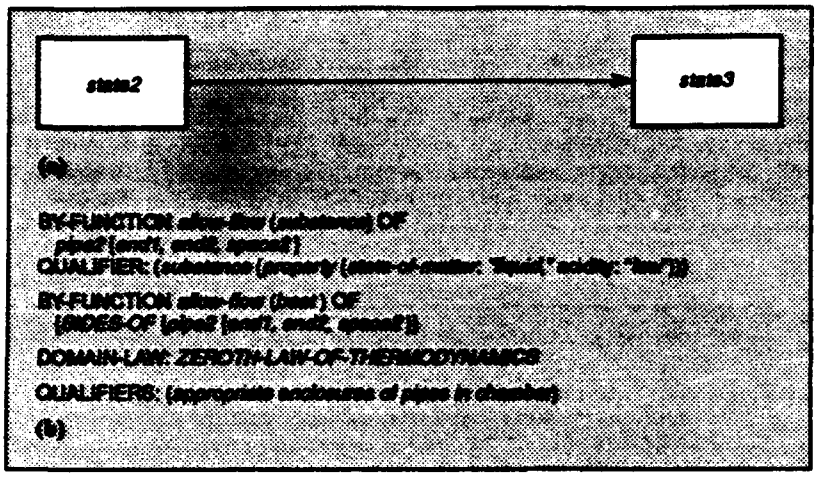


Figure 5. Annotations and qualifiers for a causal transition in nitric acid cooler: state transition (a) and annotations (b).

(2) *By-Function-Of-<component>*. In an electrical circuit, the state transition "Switch(on) → Voltage v at the terminals" might be explained by pointing to the function of the battery as a source of voltage, that is, by the annotation, "By-Function(voltage generation) Of Component(battery)." In fact, a major goal of causal explanation is to detail device behavior in terms of component properties and interconnections. Again, a large part of human domain expertise is in the forming of knowledge about generic components and their functions, even if the details of component function are unknown. The ability to explain device functions partly in terms of component functions and to explain the latter in terms of subcomponents helps form functional/component hierarchies in explanation and design.

(3) *By-Domain-Law <...>*. Another form of explanation occurs through appeal to domain laws. In the engineering domain, scientific laws provide the ultimate explanation for device behavior. For example, the state transition, "5 volts at the input → 2 amperes through the load" might be explained as "By-Domain-Law(Ohm's law: voltage = current • resistance)."

For a particular device, any realistic FR description tapers off at some level of components and CPDs. This is an example of the general incompleteness inherent in FR and DR.

The function of a device (or a component) is explained by pointing to a CPD. When we use a By-Function annotation, we are actually pointing to a CPD that partly explains that function. There are two differences between these annotations. The first one is how the CPDs are indexed. In the case of a By-CPD annotation, we are pointing to a piece of prior knowledge explicitly labeled as a causal process, while in the case of a By-Function annotation, we are pointing to prior knowledge of a component function. The second difference is that a component function may not have a CPD explicitly available to explain it. Often we know many components only by their functions.

Sometimes additional noncausal links must be added to arrive at the predicate of interest. For example, for an amplifier, we may have constructed the CPD,

> Voltage 1 at the input → ... → Voltage 10 at the output,

but the function that needs to be explained might be "amplification of 10." A noncausal definitional/abstraction link can be used to arrive at the node "amplification of 10" from "Voltage 10 at the output." Such links can be used to indicate an inference that follows from predicates in the earlier nodes.

In addition to annotations, the links may have qualifiers that state conditions under which the transition takes place. In FR, the qualifier Provided(P) indicates that condition P must hold during the causal transition for the transition to be initiated and completed, and If(P) indicates that condition P must hold at the moment the causal transition starts. The conditions can refer to the states of any component or substance. Many of these qualifiers are eventually translated as conditions on the structural parameters.

Figure 5 shows a fully annotated causal transition from "state2" to "state3" in the Nitric Acid Cooler. It uses two functional annotations and one domain-law annotation, and employs conditions on the substances and structure as qualifiers. The qualifiers include conditions on the properties of the substance (it should be a "liquid of low acidity") and structural conditions ("the chamber fully encloses pipe2"). Note that a transition may have more than one annotation or qualifier.

Functions. Every device has intended functions. Keuneke[6] has identified four types: ToMake, ToMaintain, ToPrevent, and ToControl. Formal definitions of these function types have been developed,[7] but for our purposes, the following informal ones should suffice. All the function types above except ToControl take as argument a predicate, say P_f, defined over the state variables of the device. The function is the ToMake type if the goal is to make the device reach a state in which P_f is true. After that state is reached, no specific effort is needed to keep the predicate's value to True (or it doesn't matter what state the device goes to after the desired state is reached).

A function is type ToMaintain if the intention is to take the device to the desired state and if the device must causally ensure that the predicate remains True in the presence of any external or internal disturbance that could change the device state. The function type is ToPrevent if the goal is to keep P_f from being true and some active causal process in the device must ensure it. (Logically, ToPrevent P can be written as ToMaintain (Not P), but there are important differences in practice. Pragmatically, a designer charged with ensuring that a device, for example, doesn't explode uses knowledge indexed and organized for this purpose. Preventing this explosion by using, say, a thick pipe is not the same as maintaining some dynamic state variable in a range.)

The function type ToControl takes as argument a specified relation $v_0 = f(v_1, \ldots v_n)$ between state variables v_0, $v_1, \ldots v_n$, and the intent is to maintain this relationship. That is, we wish to

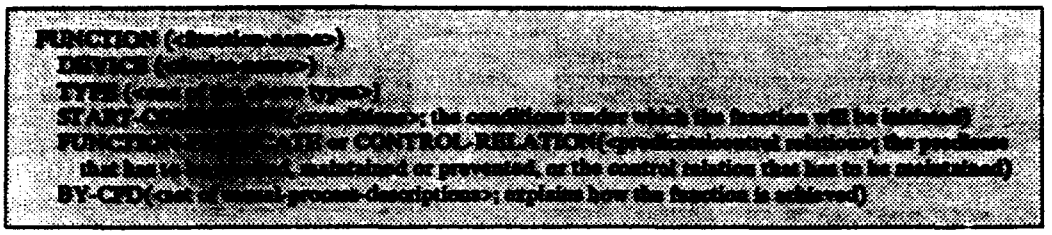


Figure 6. Elements of function specification in Functional Representation.

control the value of a specific variable as a function of the values of some other variables.

Function *F* thus has the descriptive elements shown in Figure 6. Now reconsider the example in Figure 1. Hot nitric acid goes into a heat exchanger, exchanging heat with the water that is being pumped in. The water becomes hotter while the acid becomes cooler. Figure 7 provides the functional definition of NAC. The complete FR is given by specifying the device name, its structure, state variables of interest, functions of interest, and the functional template, including the CPDs. The FR language has many implementations, each with a somewhat different syntax. We have used a composite syntax, chosen mainly for expository effectiveness. We have suppressed many details by giving English-language descriptions of the intended information within parentheses or curly brackets. For example, we say "Qualifiers: (appropriate enclosures of pipes in chamber)" in Figure 5. Goel[5] provides a detailed syntax for representing the relevant relations about pipes.

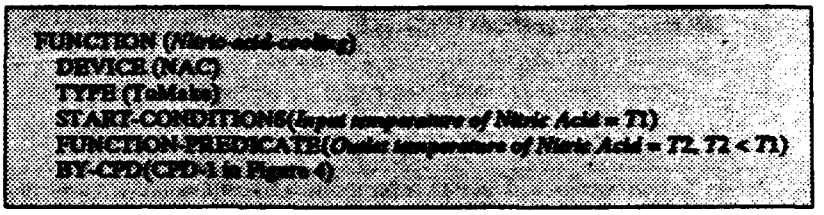


Figure 7. Function specification of nitric acid cooler.

Using FR to represent a device. A language by itself does not fully specify how it should be used; therefore, a few remarks on how FR should be used are in order. First, there is no unique FR for a device. There can be differences in the ways various agents describe how a device works. The differences might relate to how the explanation was decomposed, what background knowledge was assumed, and the intended uses of the causal explanation. Second, choices have to be made about what functions should be explicitly represented. While every device has intended functions, there are a number of implicit functions (such as a design specification of a TV set that does not explicitly record that it is not intended to explode).

Range of FR applicability. While functional representation as an idea is quite general, the set of primitives discussed so far only covers a subclass of devices. Not all devices have functions that have to be understood in terms of a temporally evolving causal process. For example, the device "chair" has the function "to support the seating of a human." However, this function is not accomplished by any causal process that involves state changes. Instead, the function is achieved by the chair's shape. Bonnet[8] has identified a function type called ToAllow to account for these instances.

Second, FR discretizes continuous causal processes into node-to-node transitions in which some predicate of interest changes at each node. There are many phenomena for which explanations are given by simply displaying the behavior of a continuous function. For example, gear teeth meshing smoothly can best be explained by showing the displacement functions of the teeth.

Third, more research is needed to represent functions having temporal relations between states that serve as an essential part of the function definition. For example, an electronic sawtooth signal generator goes through a certain series of state changes when charging and discharging its output capacitor. The function to be achieved corresponds not to the device being in a single desired state but to its repeatedly cycling through a sequence of states, with each state having a certain duration.

Representing these aspects of a causal process is an open research issue. On the other hand, there is no real constraint that the devices must be physical for useful FRs to be constructed for them. The FR framework has been used to represent computer programs and to reason about program errors.[9] FR has also served as a vehicle for reasoning about complex physical systems; references 4, 6, 7, and 10 provide some examples.

FR as design rationale

While FR does not encode available alternatives to a choice and why they were not chosen, it does provide a partial rationale for choices made about components and their configuration. We say partial because the CPD encodes only the directly causal role of a component. Of course, there could be other reasons for choosing a particular component value. Also, the role of a component in achieving a function can be distributed over several transitions and different CPDs. The rationale for why a component was chosen might actually reflect these multiple constraints. In an electrical circuit, the real explanation for why a resistance was chosen to be 1,000 ohms might be that the resistance had to be less than 2,000 ohms for a specific transition, greater than 600 for another transition, and standardized at 1,000 ohms for easy procurement. Because the latter information is not part of the causal account, it is not directly represented in FR.

We earlier identified a number of

tasks that DR should be able to support: diagnostic knowledge generation, simulation, design verification, and redesign/case-based design. FR can support many of these tasks to the degree that they require causal knowledge about device operation. Of course, all these tasks can also benefit from information that comes from other DR components. In particular, the redesign task often requires a knowledge of why certain choices were not made. Additional knowledge from other DR components could be useful for those aspects of the task. In this section, we give brief overviews of how to use FR for diagnostics generation, design verification, and redesign.

Generation of diagnostic knowledge. For simplicity, let's first consider a CPD in which each transition has only one annotation. Consider a transition in a CPD of the form

By-Function *F*-Of-Component *c*

n_1 ——————————> n_2

Suppose the device is in the partial state n_1, that is, the device is in a state that satisfies the predicates corresponding to n_1. Suppose we test the device and observe that it fails to reach n_2. What conclusions can we draw? The CPD asserts that the device goes from partial state n_1 to n_2 because of the function *F* of component *c*, and therefore, we can hypothesize that the failure to reach n_2 is due to component *c* not delivering function *F*. Corresponding to this transition, we can identify a possible malfunction state "Component *c* not delivering function F." The diagnostic rule, "device satisfies n_1 but not n_2," can be used to establish this malfunction mode.[4]

If the annotation had instead been "By-CPD CPD-1," where CPD-1 is a specific CPD, we could similarly examine CPD-1 to see why this transition failed (a transition in CPD-1 must have failed if the transition from n_1 to n_2 failed). Ultimately, we can identify the component function responsible for the failure of the device.

There is no malfunction corresponding to a transition with the annotation By-Domain-Law; a domain law cannot fail to hold. Of course, the designer's account of the role played by the domain law could be incorrect, but we are assuming here that the FR itself is correct. How to verify the FR is an interesting issue in its own right but is not the subject under discussion.

Not all diagnostic knowledge can be derived from design information alone.

The technique of identifying a component malfunction either directly from the annotation By-Function or by recursive application of By-CPD leads to a diagnostic tree with leaf nodes that are malfunctions of components or subcomponents. The diagnostic rule for each malfunction consists of rules of the form "If the predicates corresponding to node n_i are true, but those corresponding to n_j are not true, then . . . ."

What happens when we have more than one annotation, as in Figure 5, where the transition is explained by appealing to more than one function? In this example, the transition can fail when "pipe2" is blocked, or has a failure in its thermal (heat-exchange) function, or when the conditions in the qualifiers are not satisfied. In this case, the transition's failure can only identify these as possible malfunctions. It cannot establish them because additional information is necessary.

Note, however, that not all diagnostic knowledge can be derived from design information alone. For example, rank-ordering diagnostic hypotheses in terms of likelihood and pursuing them in the order of most-to-least probable is quite common in diagnostic reasoning. But this ordering requires knowledge of failure probabilities for components. This information is not derivable from a causal model of how a device works. Additional information in the form of failure rates is needed. Conversely, not all diagnostic knowledge derived from causal models is directly usable, since some variables mentioned in the diagnostic rules generated from causal models may not be directly observable. Additional inference may be required.

Design verification. By comparing the behaviors predicted by simulation and the FR of the desired device function, one can verify whether the design will achieve its functions in the intended fashion. There are two potential difficulties, however, in using FR and simulation results for design verification.

First, there is a possible gap in levels of abstraction between the language used in the function specification in FR and the language in which the simulated behavior is represented. For example, the behavior of transistors and resistors may be simulated in terms of currents and voltages, but the function of the circuit as a whole might be described as an oscillator or as an adder. To make verification possible, one must ensure that each abstract concept used in the functional specification is clearly defined in terms of the concepts employed in simulating the behavior.

Second, the model used for simulation may contain details irrelevant to verifying the function of interest, or it may not contain all the relevant information. For example, a pipe as a component can be modeled as a conduit for fluid flow, as an object with thermal properties, or as both. If the function to be verified concerns only flow, we would use this information to construct an appropriate simulation model that is as simple as possible while containing all relevant aspects.

The CPD can be used as follows in the design-verification task.[7] The predicates that appear in the CPD definitions of the nodes and the functional predicate P_f are terms of interest at the device level. These predicates are first defined in terms of objects and predicates that appear in component definitions. For example, suppose the predicate "Amplification-Level" appears in the description of a CPD node and that the component behaviors are in terms of voltages and currents. We first define the predicate in terms of voltages at the input and output of relevant components. Second, we perform the simulation. Finally, we attempt to establish that states in the simulated behavior correspond to each CPD node and that each CPD transition in fact occurs in the simulated behavior in the intended way.

Consider a transition from n_1 to n_2 in the CPD of an electrical circuit. Let's say it is annotated as By-Function *F*-Of-Component *c*. Suppose n_1 is characterized by the predicate "amplification at port $p_1 > 15$" and n_2 by "amplification at port $p_2 > 30$," where "amplification at

port x" is defined in the language used in the simulation model as

Voltage at x / Voltage at p_0.

To verify this portion of the CPD from n_1 to n_2, we search for a state in the simulated behavior where the predicted values of the voltages at p_0 and p_1 satisfy the condition for n_1, using this definition of amplification. If such a state is found, we must also find the same (or later) state in which the condition for n_2 is satisfied. If such a state is found as well, we at least know that the situations described by n_1 and n_2 actually take place in the simulated behavior.

However, before we can claim verification of the CPD transition, we must show that the realization of condition n_2 was caused by condition n_1 and function F of component c. If component c had no role in the transition, it may not have been needed.

The meaning of one event causing another is a contentious philosophical issue. However, within the context of any one modeling and simulation scheme, one can define causal relations unambiguously. Iwasaki and Chandrasekaran[7] provide such a definition of causal relations in the context of a particular model-formulation and simulation system called DME (Device Modeling Environment), based on the bottom-up, behavior-oriented approach (see sidebar again). In the verification scheme described in that reference, this definition is used to show that the conditions specified by a CPD node and by the annotations on the transition link out of the node play a causal role in achieving the condition specified by its successor node.

Redesign. In this task, the goal is to modify the artifact so that it meets somewhat different functions. As previously noted, if the required changes in function are drastic, radical structural alterations may be needed, possibly requiring another design from scratch. However, if changes are small, redesign can be accomplished by relatively simple modifications to the existing structure, perhaps by parametric changes to the components and substances. We examine the role of DR in general and FR in particular in the redesign problem, assuming that the required changes are parametric.

To accomplish the task, we need to solve three subtasks: identifying components that require modification, identifying needed modifications, and verifying that these modifications produce the desired function changes.

There is no real sense in which a design rationale can be completely represented.

Deciding on the appropriate modifications requires additional knowledge, some of which might be found in other parts of DR. For example, if DR includes the reasons certain design options were considered but not chosen, this information might be relevant to the redesign problem. Regarding verification, Pegah, Bond, and Sticklen's use[10] of FR for parametric simulation is relevant. They show how FR can be viewed as a form of compiled simulation and suggests ways in which FR can incorporate information about device behavior over ranges of component parameters. With this information, one can straightforwardly derive device behavior when component parameters are changed.

We outline how Kritik,[5] a system that performs a form of case-based design, uses FR to identify candidates for modification. Suppose we want to modify NAC to cool high-acidity sulfuric acid rather than low-acidity nitric acid. First, Kritik compares the functions desired of and delivered by the candidate NAC design and notes that they differ in

(1) the substance to be cooled (sulfuric instead of nitric acid) and
(2) a property of the substance (high instead of low acidity).

Since the substance property difference occurs in the function "cool (low-acidity) nitric acid," Kritik uses this function to access the CPD responsible for it. A fragment of this CPD, the transition from "state2" to "state3," is shown in Figure 5. Kritik traces through this CPD, checking each state transition to determine whether the goal of reducing the substance property difference ("low-acidity → high-acidity") can be achieved by modifying an element in the transition. For example, in the transition "state2 → state3," Kritik finds that "pipe2" has an "allow" function restricted to low-acidity substances.

Kritik has a typology for modifying device components:

- the parameters of a component can be tweaked,
- the modality of a component's operation can be changed, and
- one component can be replaced by another.

This typology correspondingly generates the modification hypotheses that "pipe2"

- allows the flow of high-acidity substances in a different parameter setting,
- allows the flow of high-acidity substances in a different mode of operation, and
- has to be replaced with some new "pipe2" to allow the flow of high-acidity substances.

Because how the modification is chosen does not directly relate to DR, we omit that discussion. Goel[5] details how Kritik handles the replacement of nitric acid with sulfuric acid.

Design rationale is a record of design activity: options that were considered, choices that were (and were not) made along with reasons for the decisions, and how designers satisfied themselves that the device would work as intended. We have proposed the use of a framework called Functional Representation as a candidate for capturing the latter component of design rationale. FR encodes the designer's account of the causal processes in the device that culminate in achieving its function. The representation makes explicit the components' roles in the causal process. FR has been implemented in several versions and used to represent and reason about a number of systems, but using it to represent a component of design rationale is novel.

We have discussed the limitations of FR as design rationale in that it captures only the causal component. We have also discussed the limitations of the current repertoire of representational primitives in FR for capturing the

causal component: They are restricted to devices that achieve their functions by means of causal state changes; complex temporal relations are not easy to capture. But the basic framework is extensible. The central idea that makes FR a good candidate for representing the causal component of design rationale is that it embodies a theory of how causal stories are understood. This idea will continue to form the organizing principle for any extensions of FR as well.

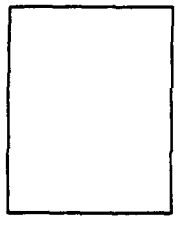
Design rationale is not only many-faceted but also intrinsically open-ended. There is no real sense in which a design rationale can be completely represented. Ultimately, much of it will appeal to shared commonsense knowledge about how objects and people behave. What aspects of the rationale to make explicit will depend on intended users and tasks. ■

Acknowledgments

Chandrasekaran's research was supported by DARPA AFOSR Contract F-49620-89-C-0110 and AFOSR grant 89-0250. Goel's work on design is supported by grants from the National Science Foundation, the Office of Naval Research Contract N00014-92-J-1234, Northern Telecom, and equipment grants and donations from IBM, NCR, and Symbolics. Iwasaki acknowledges support from DARPA and from NASA Ames Research Center (NAG2-581 and NCC2-537). We are grateful to John Josephson and the anonymous referees for many useful suggestions, and to the guest editors for their support and encouragement.

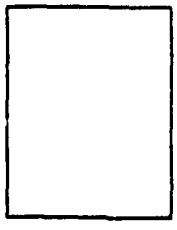
References

1. Special Issue on Design Rationale, *Human-Computer Interaction*, Vol. 6, Nos. 3 and 4, 1991.
2. A. MacLean et al., "Questions, Options, and Criteria: Elements of Design Space Analysis," *Human-Computer Interaction*, Vol. 6, Nos. 3 and 4, 1991, pp. 201-250.
3. J. Lee and K-Y. Lai, "What's in a Design Rationale?" *Human-Computer Interaction*, Vol. 6, Nos. 3 and 4, 1991, pp. 250-280.
4. V. Sembugamoorthy and B. Chandrasekaran, "Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems," in *Experience, Memory, and Reasoning*, J.L. Kolodner and C.K. Riesbeck, eds., Lawrence Erlbaum, Hillsdale, N.J., 1986, pp. 47-73.
5. A.K. Goel, "A Model-Based Approach to Case Adaptation," *Proc. 13th Ann. Conf. Cognitive Science Soc.*, Lawrence Erlbaum, Hillsdale, N.J., 1991, pp. 143-148.
6. A. Keuneke, "Device Representation: The Significance of Functional Knowledge," *IEEE Expert*, Vol. 6, No. 2, Apr. 1991, pp. 22-25.
7. Y. Iwasaki and B. Chandrasekaran, "Design Verification through Function- and Behavior-Oriented Representations: Bridging the Gap between Function and Behavior," in *Artificial Intelligence in Design*, J.S. Gero, ed., Kluwer Academic Publishers, Boston, 1992, pp. 597-616.
8. J.C. Bonnet, "Towards Formal Representation of Device Functionality," Tech. Report 92-54, Knowledge Systems Laboratory, Stanford Univ., Stanford, Calif., 1992.
9. D. Allemang, "Using Functional Models in Automatic Debugging," *IEEE Expert*, Vol. 6, No. 6, Dec. 1991, pp. 13-18.
10. M. Pegah, W.E. Bond, and J. Sticklen, "Representing and Reasoning about the Fuel System of the McDonnell Douglas F/A-18 from a Functional Perspective," submitted to *IEEE Expert*.



B. Chandrasekaran directs the Laboratory for AI Research and is a professor of computer and information science at Ohio State University. His research interests include knowledge-based systems, using images in problem solving, and the foundations of cognitive science and AI.

Chandrasekaran received the PhD degree from the University of Pennsylvania in 1967. He is editor-in-chief of *IEEE Expert*, a fellow of the IEEE and the American Association for Artificial Intelligence, and a member of the IEEE Computer Society.



Ashok K. Goel is an assistant professor with the College of Computing at the Georgia Institute of Technology. His research interests include design problem-solving and computer-aided design, mental models and model-based reasoning, and case-based reasoning and learning.

Goel received the MS degree in physics and the PhD degree in computer and information science from Ohio State University. He has served on several conference program committees, including AAAI-92. He chairs the IEEE Systems, Man, and Cybernetics Technical Committee on AI and is a 1992-93 Lilly Teaching Fellow at the Georgia Institute of Technology. He is a member of the IEEE Computer Society.

Yumi Iwasaki is a research scientist at Stanford University's Knowledge Systems Laboratory. Her research interests include model-based reasoning and reasoning with pictorial representations.

Iwasaki received the BA degree in mathematics from Oberlin College, the MS in AI from Stanford University, and the PhD degree in computer science from Carnegie Mellon University. She is a member of AAAI, the Japanese Society for AI, Computer Professionals for Social Responsibility, and Sigma Xi.

Readers can contact Chandrasekaran at the Laboratory for AI Research, 2036 Neil Ave., Ohio State University, Columbus, OH 43210; chandra@cis.ohio-state.edu.

Explanations in Knowledge Systems

The Role of Explicit Representation of Design Knowledge

B. Chandrasekaran, Ohio State University
William Swartout, University of Southern California

THE FOLLOWING TWO ARTICLES examine explanation in expert systems. The unifying idea in these projects is of general importance to explanation: The *more explicitly we represent the knowledge* underlying a system's design, the better its explanations.

Knowledge and methods of using knowledge are the fundamental elements of knowledge systems. Much knowledge-system research has been concerned with developing *increasingly explicit representations of these* elements to support increasingly sophisticated techniques for knowledge acquisition, system building, and explanation. From an explanation standpoint, explicit representations of knowledge and method enable a knowledge system to examine its own structure and produce explanations from the same structures used for reasoning.

The idea that explicit representations facilitate explanation was recognized early on (for example, in Mycin[1]). It soon became evident that higher level strategies played a role in a knowledge system's ability to solve problems. Researchers began to explicitly represent problem-solving strategies (methods or plans) for using knowledge to solve problems. Examples here include Digitalis Advisor,[2] MDX,[3] and Neomycin.[4]

EXPLANATIONS OF A KNOWLEDGE SYSTEM'S CONCLUSIONS CAN BE AS IMPORTANT AS THE CONCLUSIONS THEMSELVES. THE UNIFYING IDEA IN THE NEXT TWO ARTICLES IS OF GENERAL IMPORTANCE TO EXPLANATION: THE MORE EXPLICITLY WE REPRESENT THE KNOWLEDGE UNDERLYING A SYSTEM'S DESIGN, THE BETTER ITS EXPLANATIONS.

Another important idea for knowledge systems is that we can obtain problem-solving knowledge from other, more general knowledge. In justifying its conclusions, a knowledge system might need to justify the knowledge used to reach them. This in turn requires access to the more general knowledge that produced the system's knowledge. (Although such general knowledge is sometimes called "deep" knowledge, or "first principles," it is not better knowledge; it is simply more general, that is, useful for more purposes rather than for a specific problem type.) Knowledge systems based on explicit representations of knowledge and method, with information about how and from what their knowledge was obtained, are the foundation for producing good explanations.

Each type of explicit knowledge makes specific kinds of explanation possible. Also, each such representation makes explicit an aspect of the design of the knowledge system itself. For example, representing generic problem-solving methods or strategies is a way to make explicit the strategies that are often implicit in expert-system knowledge bases. Similarly, representing the more general knowledge used to derive specific pieces of knowledge in the knowledge base involves making another aspect of the design explicit, namely, where the knowledge in the knowledge base came from. Thus, the operational principle at work here calls for increasingly explicit design knowledge.

JUNE 1991 0885/9000/91/0600-0047 $1.00 © 1991 IEEE

Tasks and knowledge

Let's say system S performs a task T using knowledge K. (We would normally use S_T and K_T to indicate that S is a system that solves T, and that K is the knowledge related to T, but we dispense with the additional notational complexity here.) For example, Mycin is a problem-solving system (S) that performs the task of consulting about infectious diseases (T) using the rules in its knowledge base (K). A task is a collection of problem instances of a certain type. For example, Mycin can solve a number of instances of consultation problems in infectious diseases.

$\Delta(S)$ is the knowledge needed to design S, and $R(S)$ is the design record (the record of how $\Delta(S)$ was used to design S). $\Delta(S)$ typically includes substantial knowledge about the domain's subject matter, the nature of the task, the range of appropriate strategies, the architecture of S, how the parts of S accomplish T, and the origin and range of applicability of K. $R(S)$ would consist of the various design documents recording the design process. In the Mycin example, Δ(Mycin) is the designer's knowledge about the domain, the task, and AI architectures (most of which is never made explicit), while R(Mycin) explicitly captures a small part of its design using an informal notation (such as English or diagrams). Abstractly, knowledge-system design is a process that produces S from $\Delta(S)$, and $R(S)$ as a partial record of this process.

By the nature of design knowledge, $\Delta(S)$ can support the design of a range of systems of which S is a specific instance. For example, $\Delta(S)$ might contain a parametric family of strategies, one of which is instantiated in S. In the Mycin example, the same underlying knowledge of the domain, task, and strategies could be used to design variants of Mycin that perform different versions of the task or the same task in different ways.

Much expert-system research has emphasized the advantages of explicitly representing K. In fact, expert systems as a field is identified by its explicit representation of K and its application of various forms of inference to K to solve problems. While we need not explicitly represent $\Delta(S)$ to solve T, there are several advantages in explicitly representing relevant components of $\Delta(S)$, including how knowledge in them was used:

- Knowledge in $\Delta(S)$ and $R(S)$ about strategies and about K lets us explain the behavior of S and justify its conclusions.
- By operating on the representation of strategies in $\Delta(S)$, we can generate a range of behaviors of S.
- The explicit representation in $\Delta(S)$ of knowledge and strategies used by S makes system maintenance easier.

Of course $\Delta(S)$ is open-ended, so we cannot explicitly and formally represent all of it. However, as research in knowledge types, strategies, and architectures progresses, we will have an increasingly rich vocabulary to represent more and more parts of $\Delta(S)$.

EXPLICIT REPRESENTATIONS OF KNOWLEDGE AND METHOD ENABLE A KNOWLEDGE SYSTEM TO EXAMINE ITS OWN STRUCTURE AND PRODUCE EXPLANATIONS FROM THE SAME STRUCTURES USED FOR REASONING.

The accompanying articles

The following articles present results from two groups, both concerned with using knowledge about a system's design to explain it. In the first article, Michael C. Tanner and Anne M. Keuneke report on work at Ohio State University, which has concentrated on

- identifying appropriate strategy abstractions (the generic-task work);
- exploring the relation between task requirements and strategies available in S; and
- understanding the relationship between diagnostic-task knowledge and structure-function models of the device being diagnosed.

The work reported in the article focuses on producing strategic and task explanations and on justifying knowledge.

The second article comes from the Explainable Expert Systems project,[3] in which William Swartout and his associates have focused on

- distinguishing and providing explicit representations for several kinds of knowledge in $\Delta(S)$, including a domain model and general strategies;
- using an automatic program writer to create an explicit design record $R(S)$ that records how knowledge in $\Delta(S)$ was applied to create S; and
- capturing the "design" of explanations, that is, what the system was trying to say and how it was trying to say it.

In the terms used above, the EES project is concerned with two main tasks: the task T solved by the knowledge system S, and the task of constructing explanations $E(T)$ of S's performance. Let $\Delta(E(T))$ denote the design knowledge that supports the construction of the explanation $E(T)$, and let $R(E(T))$ be the record of how that design knowledge was used to construct a particular explanation. $R(E(T))$ captures what the explanation component tried to convey in a particular explanation, what explanation strategies it used to convey it, and what alternative strategies exist to get the same points across. This is exactly the information needed by the dialogue component to understand follow-on questions in context and to correct misunderstandings. In their article, Swartout, Cécile Paris, and Johanna Moore discuss knowledge justifications and issues related to presenting explanations and dialogue with users.

Acknowledgments

This work has been supported by the Air Force Office of Scientific Research (grant 89-0250), by the Defense Advanced Research Projects Agency (grant MDA 903-81-C-0335 and contracts F30602-85-C-0010 and F49620-89-C-0110), and by a NASA Ames cooperative agreement (number NCC 2-520). We thank Saul Amarel, Cécile Paris, and Michael Tanner for their comments on earlier drafts.

References

1. E.H. Shortliffe. *Computer-Based Medical Consultations: Mycin*. American Elsevier, New York, 1976.
2. W.R. Swartout. "A Digitalis Therapy Advisor with Explanations." *Proc. Fifth Int'l Joint Conf. Artificial Intelligence*

(IJCAI 77). Morgan Kaufmann, San Mateo, Calif., pp. 819-825.

3. B. Chandrasekaran and S. Mittal, "Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and Related Systems," in *Advances in Computers*, Vol. 22, M.C. Yovits, ed., Academic Press, New York, 1983, pp. 217-293.
4. W.J. Clancey and R. Letsinger, "Neomycin: Reconfiguring a Rule-Based Expert System for Application to Teaching," in *Readings in Medical Artificial Intelligence*, W.J. Clancey and E.H. Shortliffe, eds., Addison-Wesley, Reading, Mass., 1984, pp. 361-381.
5. B. Chandrasekaran, "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," *IEEE Expert*, Vol. 1, No. 3, Fall 1986, pp. 23-30.
6. W.R. Swartout and S.W. Smoliar, "On Making Expert Systems More Like Experts," *Expert Systems*, Vol. 4, No. 3, 1987, pp. 196-207.
7. J.D. Moore and W.R. Swartout, "Pointing: A Way Toward Explanation Dialogue," *Proc. Eighth Nat'l Conf. Artificial Intelligence (AAAI 90)*, MIT Press, Cambridge, Mass., pp. 457-464.
8. W.R. Swartout, "Knowledge Needed for Expert System Explanation," *Future Computing Systems*, Vol. 1, No.2, 1986, pp. 99-114.

B. Chandrasekaran directs the AI group and is professor of computer and information science at Ohio State University. He is also editor-in-chief of *IEEE Expert*. His research interests address knowledge-based reasoning. Chandrasekaran received his BE with honors from Madras University in 1963 and his PhD from the University of Pennsylvania in 1967. Readers can reach him at 217 Bolz Hall, Ohio State University, 2036 Neil Ave., Columbus, Ohio, 43210; e-mail, chandra@cis.ohio-state.edu

William Swartout is director of the Intelligent Systems Division and an associate research professor of computer science at the USC's Information Sciences Institute. His research interests include explanation and text separation. He started and led USC/ISI's Explainable Expert Systems project, and he was the principal designer of the Xplain system at MIT. Swartout received his bachelor's degree from Stanford University, and his MS and PhD in computer science from MIT. Readers can reach him at USC/ISI, 4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292-6695.

Appears in Artificial Intelligence in Design '92, J.S. Gero (ed.)
Kluwer Academic Publisher, 1992, pp. 597-616

DESIGN VERIFICATION THROUGH FUNCTION- AND BEHAVIOR-ORIENTED REPRESENTATIONS:

Bridging the gap between function and behavior

Y. IWASAKI
Knowledge Systems Laboratory
Stanford University
701 Welch Rd., Palo Alto, CA 94304

and

B. CHANDRASEKARAN
Laboratory for AI Research
Dept. of Computer & Information Science
The Ohio State University
217 B, Bolz Hall
2036 Neil Avenue
Columbus, OH 43210-1277

Abstract. This paper focuses on the task of design verification using both knowledge of the structure of a device and its intended functions. In particular, it addresses the question of when one can say a behavior predicted by a prediction system achieves the desired function in the manner intended by the designer. We use Functional Representation (Sembugamoorthy & Chandrasekaran 1986) to represent the function of a device and the expected causal mechanism for achieving it. We present a formal definition of matching between a system trajectory generated by a simulation system and the description of a causal process to achieve a function expressed in Functional Representation. We demonstrate behavior verification based on the definition, using two predicted behaviors of the electrical power system of a satellite. We believe that evaluating a behavior with respect to the expected causal process as well as the function improves the chances of uncovering hidden flaws in a design that may otherwise go undetected at an early stage.

1. Introduction

Simulation of the behavior of the design of a structure is an important means for design evaluation, which must ascertain that the design achieves the intended function. To achieve a robust modeling and simulation capability, a system must be able to compose a simulation model from pieces each of which may be applicable to a variety of situations. At the same time, in order

to provide a useful feedback about the design of a device based on the result of simulation, the system must be able to evaluate the predicted behavior with respect to the knowledge of the function.

In this paper, we focus on the task of design verification using both knowledge of the structure of a device and its intended functions. In particular, we address the question of when one can say a behavior predicted by a prediction system achieves the desired function in the manner intended by the designer. We use Functional Representation (Sembugamoorthy & Chandrasekaran 1986) to represent the function of a device and the expected causal process for achieving the function. We will present a formal definition of matching between an expected behavior and a predicted behavior. Finally, we will demonstrate behavior verification based on the definition, using an actual example of behavior predicted by a simulation system.

1.1 BEHAVIOR-ORIENTED AND FUNCTION-ORIENTED APPROACHES TO MODELING

Research in model-based reasoning about physical systems has emphasized representation of structures and reasoning about behavior from the knowledge of their structures and physical principles. Several model-based reasoning systems have been built (Falkenhainer & Forbus 1991, Crawford et al. 1990, Iwasaki & Low 1991) that formulates a model of a device based on its structure and predict its behavior. An important requirement in the approach taken in these systems, which we shall call the *behavior-oriented* approach, is that the knowledge is stored in small pieces, each representing a conceptually independent physical phenomenon such as a physical process or an aspect of the behavior of a component. For the pieces to be composable, each of them must be defined in a context-independent manner as much as possible in the sense that there is no unstated assumption about the surroundings of a component or the function of the whole device. These systems predict a behavior in terms of a sequence or a graph of states, each of which is characterized by the set of applicable knowledge pieces, implied constraints, and variable values.

This type of model-based reasoning capability is useful for a system aimed to help in design, since it allows the system to formulate a behavior model automatically and to simulate its behavior so that the designer can discover behavioral implications of design decisions easily. However, an account of behavior in the form of a sequence of states must be evaluated to be useful for further development of the design. Does the predicted behavior achieve the desired function? Does it do so in the way the designer intended? These are crucial questions in providing a useful feedback to the designer. In order for a model-based reasoning system to answer such

questions, it must have knowledge of the function of the device -- WHAT it is supposed to do -- and the expected behavior -- HOW it is supposed to achieve the function.

Functional Representation (FR) is a representational scheme for the functions and expected behavior of a device. FR represents knowledge about devices in terms of the functions that the entire device is supposed to achieve and also of the sequence of causal interactions among components that lead to achievement of the functions. FR takes a top-down approach to representing a device in contrast to the bottom-up approach of behavior-oriented knowledge representation and reasoning schemes. In Functional Representation, the function of the overall device is described first and the behavior of each component is described in terms of how it contributes to the function, while in a behavior-oriented approach, the behavior of the entire device is inferred from those of individual components.

In order to evaluate a design, one must be able to predict the possible behavior of the design, as well as to determine whether the predicted behavior achieves the expected functionality. Verification that a behavior of a designed artifact achieves the desired goal must ascertain the following:

(1) the overall function of the device is achieved,
(2) the expected chain of events happen in the predicted behavior, and
(3) the causal connections expected between events exist in the predicted behavior.

The purpose of this paper is to investigate this concept of behavior verification and provide a formal definition of behavior verification of a design with respect to its intended functions and the expected causal processes for achieving the functions. As an example of a model-based reasoning system, we use DME (Device Modeling Environment) developed at Stanford University (Iwasaki & Low 1991). Given a design of a device, DME formulates a computational model and predicts its behavior.

This paper is organized as follows: In Section 1.2, we will briefly describe DME. In Section 2, we formally define functions, expected behavior, and what it means for a predicted behavior to match an expected behavior. Section 3 presents an example of behavior verification. We conclude by discussing future work and related work in Section 4.

1.2 DEVICE MODELING ENVIRONMENT

DME is a program developed by How Things Work project (Fikes et al. 1991). The goal of the project is to provide a computational environment for design of electromechanical devices, and DME is the device modeling program which forms the core of the environment. Given the topological

description of a device and initial conditions, DME formulates a mathematical model and simulates its behavior.

In DME, knowledge about physical phenomena is organized into *model fragments* in the knowledge base. Each model fragment represents knowledge of a conceptually distinct physical phenomenon such as a physical process, component behavior characteristics, etc. DME takes an input description of the initial state, including the topological model of the device, and searches the knowledge base for model fragments that are applicable to the given situation. Equations to describe the behavior of the device are formulated from the constraints associated with the set of model fragments thus found. The equations are used to predict the behavior of the device qualitatively using QSIM (Kuipers 1986) or numerically. During prediction, if there are any changes in the set of applicable model fragments, the set of equations is updated accordingly and prediction continues with the new equation model.

Some model fragments represent instantaneous changes, which are phenomena that take place too quickly to model as continuous phenomena. Such model fragments do not have constraints but they have consequences, which are facts to be asserted. When an instantaneous model fragment becomes active, a new state is generated immediately to follow the current state, and the consequences are asserted in the new state.

A model fragment m can be interpreted as one large implication of the form $P_m \Rightarrow E_m$ or $P_m \Rightarrow R_m$, where P_m, E_m, and R_m denote respectively the conditions for the applicability of the model fragment, the behavior constraints (equations in the case of a continuous phenomenon), and the consequences of m (in the case of a discontinuous phenomenon).

Definition 1. A device state is represented as a set of state variables $\{V_S\}$ consisting of values of all the variables of interest in the description of the device. State variables can be either continuous or discrete.

Definition 2. A device trajectory, T_r, represents the course of behavior of the device over time. It is a linear sequence of states.

2. What does it mean to verify that a design achieves an expected function?

In this section, we define what it means for such a simulated behavior to achieve the expected behavior represented in FR. This requires introduction of the notions of a *causal process description* (CPD) and a *function* in FR. A CPD is a causal explanation of how certain states of interest come about by exhibiting a sequence of causal transitions. The transitions are annotated by different types of causal explanation.

Definition 3. A Causal Process Description (CPD) is defined as a pair *{C, G}*, where *C* is the applicability condition and *G* is a directed graph *G = {N, L}*. *C* specifies the condition under which the device is expected to behave as specified by *G*. *C* is a necessary condition for applicability of CPD but not a sufficient condition. *N* is a set of nodes and *L* is a set of directed links among nodes. Each node represents a partial description of a state. There are two distinguished nodes in *N*, the initial node, N_{init}, and the final node, N_{fin}. Each link represents a causal connection between nodes. The graph may be cyclic, but there must be a directed path from N_{init} to N_{fin}.

A link may have an attached *qualifier*, *By-function-<f>-of(c)*, where *c* is a component, to indicate the conditions under which the transition will take place. A link can also have *annotations* of the types, *Provide(p)*, *If(p)* and *Trigger(p)*, where *p* is a wff, to indicate the type of the causal explanation to account for the transition.

In order for us to be able to relate a *Tr* and a CPD, we require that *each node in a CPD must be given a definition in the form of a wff about objects and predicates defined in terms of model fragments attributes.* We let *def(n)* denote such a definition of a node *n*. For example, the node "Battery charging" is defined as $dC/dt > 0$, where *C* is the variable, charge-level of the battery. With such a definition, a node in a CPD becomes a partial description of a state in *Tr* using attributes defined in the model fragment library.

Definition 3 mentions qualifiers and annotations that can be attached to the links between the nodes in CPD. The full list of proposed annotations can be found in (Sembugamoorthy & Chandrasekaran 1986) and (Keuneke 1991). For a link from n_i to n_j, an annotation *By-function-<f>-of(c)* means the causal interactions going from n_i to n_j must involve *c* achieving its junction. The purpose of a qualification is to allow a causal transition to be explained in further detail by CPD's of component *c*. In contrast, qualifiers allows one to specify further conditions on the causal transition. *Provided(p)* means that condition *p* must hold during the causal transition. *If(p)* means that the condition *p* must hold at state n_i. *Trigger(p)* means that *p* must not hold before n_i, but must hold at some point after n_i (inclusive).

In summary, a CPD describes a causal process from some perspective at the device level, and the conditions on the causal transitions and explanations of them are given as part of the description. Figures 1 and 2 show examples of CPD's. They are for the electrical power system (EPS) aboard a satellite, which we will use in Section 3 as an example. N_{int} and N_{fin} in each CPD are indicated by a box in dashed lines and a box in thick lines respectively.

Definition 4: A function F is defined as a quintuple $\{Type_F, P_F, Dev_F, C_F, G_F\}$, where

- $Type_F$: One of {*ToMake, ToMaintain, ToPrevent, ToControl*}.
- P_F: The functional goal, i.e. the wff that the function is to make true.
- Dev_F: The device that this function is a function of. This has to be a model fragment in the DME's knowledge base.
- C_F: The condition which specifies when the function must be achieved.
- G_F: The set of CPD's describing the causal mechanism to achieve the function.

Figure 3 shows the function of EPS. We consider four types of functions; *ToMake, ToMaintain, ToPrevent,* and *ToControl* (Keuneke 1991). Note that a device can have multiple functions, in which case each function will be represented separately. In case of a function of type *ToControl*, P_F must be of the following form:

$(= v_0\ f(v_1 \ldots v_n))$,

where v_i's are variables and f is some function of its arguments.

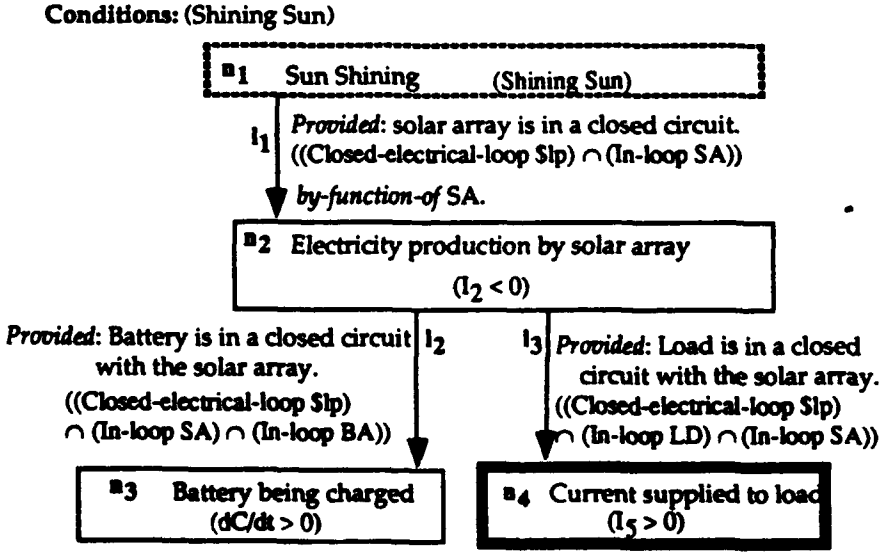


Figure 1: CPD_1 of EPS

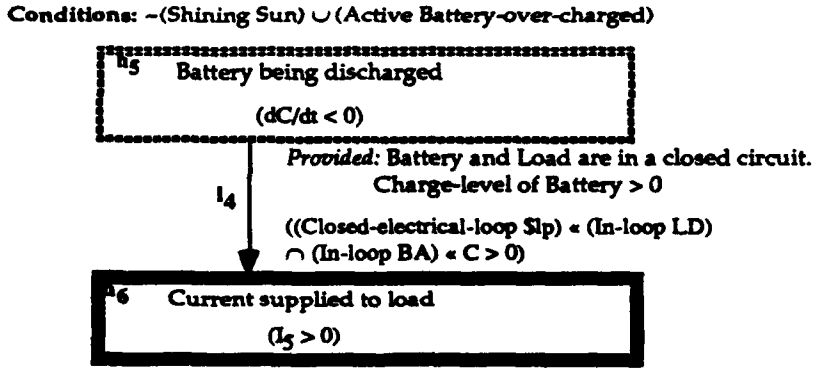


Figure 2: CPD 2 of EPS

Given a device description and initial conditions, we can generate a *Tr*. Suppose we also have an intended function for the device and associated CPD's. Intuitively, we would like to say that the device achieves the function in *Tr* if (1) the functional goal is achieved, (2) there are states in *Tr* matching all the nodes in the CPD in the specified temporal order, and (3) for each causal link in the CPD, there is a causal path in *Tr* that connects the cause to the effect. In order to make these conditions more precise, we must define the concept of a causal path in a trajectory. Then, we will define what it means for a trajectory to match a CPD.

For the rest of this paper, we use the following notation: We will attach *[s]* to wff's, model fragments and variable to denote the following axioms:

p[s]: A wff *p* holds in the state *s*.

m[s]: The phenomenon represented by *m* is active in *s* .

v[s]: A wff that asserts the value of *v* holds in *s*.

We will use notations such as $<$, $>$, $\leq$, and $\geq$ to express ordering among nodes in a CPD and states in a trajectory. We write "$n_1 < n_2$" where n_1 and n_2 are nodes in a CPD to indicate that n_1 is strictly causally upstream of n_2. For states s_1 and s_2 in a trajectory, "$s_1 < s_2$" means that s_1 strictly precedes s_2 in time. Note that the ordering is partial for nodes because a node can have multiple incoming and outgoing nodes. Ordering is total for states because a trajectory is a linear sequence of states.

Function

Type$_F$: *ToMaintain*	P_F: (Powered Load) C_F : T
Dev$_F$: EPS	G_F: CPD_1, CPD_2

Figure 3: The function of EPS

2.1 CAUSAL DEPENDENCY IN A TRAJECTORY

We now present the definition of a causal dependency relation between axioms p_1 and p_2 in a trajectory, Tr. Intuitively, we say p_2 is causally dependent on p_1, written "$p_1 \Rightarrow_c p_2$", when it can be shown in Tr that p_1 being true eventually leads to p_2 being true in Tr. Before we define the causal dependency relation among wff's more precisely, we introduce the notion of *causal ordering* among state variable.

2.1.1 *Causal Ordering Among Variables.* Suppose we are given a system of variables, and suppose some set of equations relate the values of these variables. Equations by themselves are inherently acausal and symmetric, but even when people represent the behavior of a system by a set of equations, they often perceive directed causal relations among variables. Causal ordering theory (Iwasaki & Simon 86) is used to reveal causal dependencies among variables in a set of equations and produce a graph structure that encodes these relations. In order to apply the procedure, one must have a set of independent equations, each of which represents a conceptually distinct mechanism in the situation. One must also know the variables which are externally controlled. Such variables are called *exogenous* variables.

Given a set of N equations which satisfy these requirements, the first step of the causal ordering procedure is to isolate all the subsets of variables whose values can be determined independently of the remaining variables. Such a subset of variables can be found by identifying a set of n equations which contains exactly n variables but which itself does not include a proper subset containing the same number of equations as variables. Such subset is called a *minimal complete subset.* The variables in any minimal complete subset are the "uncaused causes" of the system, and they are causally independent of other variables. Each exogenous variable constitutes a minimal complete subset.

Next, the equations in all minimal complete subsets are removed from the original set of equations and their variables are also removed from the remaining equations, producing a reduced set of $N - m$ equations in $N - m$ variables, where m is the total number of equations (and variables) in all the minimal complete subsets. Then, a new independent subset of variables is determined in the reduced set. This process repeats until the set can no longer be reduced. For each equation in the original set, the variable that was reduced last is said to be *causally dependent upon* all the other variables in the equation, and a directed graph can be generated to depict the causal dependency structure of the entire set, with nodes representing variables and links representing causal dependency relations among them. Also, for each variable v in a minimal complete subset, we define $D(v)$ to be the set of all

equations in the set. In other words, $D(v)$ is the set of all the equations that directly determine the value of v.

We will write $v_1 \rightarrow_c v_2$ when v_2 is causally dependent upon v_1 according to the definition of causal ordering. When a minimal complete subset consists of more than one variable, the causal ordering procedure does not impose ordering among them since such a situation indicates the existence of a feedback loop among them. In such a case, we say the variables are *interdependent* and write $v_1 \leftrightarrow_c v_2$.

Even though the above description is given in terms of variables and equations, which imply domains of continuous variables and function, the concept applies to domains of continuous as well as discrete variables. The "equations" in the case of discrete variables can be any axiom that can be used to determine the value of one variable depending on other variables, as long as such an axiom represents some conceptual mechanism in the situation. For example, the control of a sprinkler system that turns on between 1 and 2 am every day can be represented by an axiom.

$1 \leq time \leq 2 \Rightarrow$ (On Sprinkler).

2.1.2 *Causal Dependency Relations.* Given the causal ordering procedure we can now proceed to defining causal dependencies among wff's.

Definition 5. The causal dependency relation, denoted $a_i \Rightarrow_c a_j$, is defined between two wff's, a_i and a_j, in the descriptions of states in a trajectory Tr. We write "$a_i \Rightarrow_c a_j$" and say "a_j *depends on* a_i" or "a_i *causes* a_j." The relation $\Rightarrow_c$ is transitive.

The following conditions specify when a wff can be said to be causally dependent on another in Tr:

(a) If $p[s_0], p[s_1] \ldots p[s]$ for all states from s_0 up to s, (in other words, p was part of the initial conditions and never changed), we say that $p[s]$ is exogenous, and write $\phi \Rightarrow_c p[s]$.

(b) If there exists a state $s_j < s$ such that, $\neg p[s_j]$, and $p[s_{j+1}]$, where s_{j+1} is the immediate successor of s_j in Tr, and there exists $m[s_j]$, such that $p \in R_m$, and $p[s_i]$ for all s_i between s_{j+1} and s inclusive (in other words, p becomes true at some point before s as a consequence of the phenomenon represented by m.), we say $m[s_j] \Rightarrow_c p[s]$.

(c) For each $p \in P_m$, we say $p[s] \Rightarrow_c m[s]$. In other words, for each phenomenon active in s, we say that the phenomenon being active is dependent on its precondition being satisfied.

(d) If $v_1 \rightarrow_c v_2$ according to the definition in Section 2.1.1, we say $v_1[s] \Rightarrow_c v_2[s]$.

(e) For each equation e in $D(v)$ for a variable v and a phenomenon m such that $e \in E_m$, we say $m[s] \Rightarrow_c v_2[s]$. In other words, we say v

depends on the phenomenon giving rise to the causal relation between v and whatever other variables v depends on.

(f) $v2[s] \Rightarrow_C v1[s]$ and $v1[s] \Rightarrow_C v2[s]$ if $v2 \leftrightarrow_c v1$ in the causal ordering in s.

(g) $v'[s_1] \Rightarrow_C v[s_2]$, where s_2 is the state immediately following s_1, and v' the time-derivative of v in s_1.

2.2 When is a function achieved?

We listed in Definition 4 different types of functions of devices and components. In this subsection, we spell out the conditions under which a device is said to achieve each type of function in a trajectory.

Definition 6: Let s_z denote the final state in Tr, and Dev denote either Dev_F or one of its components. A function F is said to be achieved in a trajectory Tr in any of the following cases depending on $Type_F$. In all the cases C_F must hold in the initial state of Tr. In the following,

Case 1: When $Type_F = ToMake$, F is achieved by Tr if

1.1 P_F, the functional goal, holds in the final state s_z. We denote this by $P_F[s_z]$. And,

1.2 There is some device variable v, and some state s in Tr, such that $v(s) \Rightarrow_C P_F[s_z]$ *(i.e.* this fact causally depends on the operation of the device).

Case 2: When $Type_F = ToMaintain$, F is achieved in Tr if in all states s_i in Tr, the following is true: For some s_j such that $s_j \leq s_i$ in Tr,

2.1 $P_F[s_i]$, and

2.2 There is some device variable v such that $v[s_i] \Rightarrow_C P_F[s_i]$.

Case 3: When $Type_F = ToControl$, F is achieved in Tr if

3.1 $v_0[s_z] = f(v_1[s_z], \ldots v_n[s_z])$ *(i.e.* the functional relation holds between the value of the controlled variable and the values of the controlling variables in the final state),

3.2 $v_i[s_z] \Rightarrow_C v_0[s_z]$ for $1 \leq i \leq n$ *(i.e.* the value of the controlled variable in the final state causally depends on the controlling variables), and

3.3 There is some device variable v and some state s in Tr, such that $v(s) \Rightarrow_C v_0[s_z]$

Case 4: When $Type_F = ToPrevent$, F is achieved in Tr if $\sim P_F[s]$ for any state s in Tr *(i.e. F* is achieved in Tr if the functional goal of F does not hold in any state). We make the closed world assumption that $\sim p$ unless p is explicitly known to hold.

Definition 7: A trajectory Tr is said to match a CPD if there is a mapping st from nodes in the CPD to the states in Tr that satisfies the following conditions:

1. for each node n in the CPD there is a state $st(n)$ in Tr where $def(n)$ holds, and
2. for any nodes n_1 and n_2 in the CPD, $st(n_1) \leq st(n_2)$ iff $n_1 < n_2$, and
3. for each causal link l from n_1 to n_2, there is a causal path $def(n_1)[st(n_1)] \Rightarrow_c def(n_2)[st(n_2)]$ in Tr, where $def(n_i)[st(n)]$ denotes that $def(n)$ holds in the state $st(n)$. Furthermore, if l has an attached qualifier, *Provided(p)*, p must hold for all states between $st(n_1)$ and $st(n_2)$ inclusive. If l has an attached annotation, *By-function-of*, which points to a component o, there must be a causal path $o[s] \Rightarrow_c def(n_2)[st(n_2)]$ for some state s such that $st(n_1) \leq s \leq st(n_2)$.

Clause 1 of the above definition ensures that for each node in the CPD, there is a state in Tr that matches it. Clause 2 makes sure that the temporal ordering of causes and effects in the CPD is preserved in the temporal ordering of their corresponding states in Tr. Finally, Clause 3 ensures that the causal paths exist in Tr that correspond to the causal links in the CPD.

Armed with the Definitions 1 through 7, we are now ready to state precisely what we mean by verification that a predicted behavior achieves the expected behavior.

Definition 8: We say that a trajectory Tr of a device achieves the expected behavior with respect to a function F when the following conditions are met. Tr_i denotes the subsequence of Tr from the initial state up to and including state s_i:

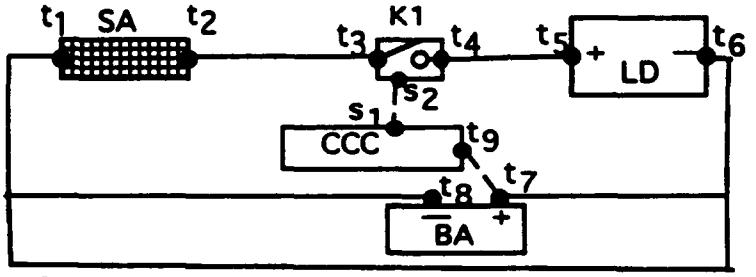
1. F is satisfied in Tr according to Definition 6, and
2. F is achieved in the expected manner, which is verified as
   - Case 1: if *TypeF* is not *ToMaintain*, Tr matches one of the CPD's of F according to Definition 7,
   - Case 2: if *TypeF* is *ToMaintain*, for each state s in Tr, a match between Tr_i and one of the CPD's exists such that $s_i = st(N_{fin})$ for the final node N_{fin} of the CPD.

Clause 1 of this definition makes sure that the function is achieved in the trajectory. Clauses 2 ensures that the function is achieved in the way the designer intended. We must distinguish the cases where the type of the function is *ToMaintain* and others, because if the function is to make or prevent some condition, we need only to show that the condition is brought about (or prevented) in the intended manner. However, if the function is to

maintain some condition throughout the trajectory, we must show that the condition is in fact brought about in the intended manner for every state.

3. Example: EPS behavior

In this section, we demonstrate behavior verification with an example of the electrical power system (EPS) aboard a satellite orbiting the earth (LMSC 1985). A simplified schematic diagram of the EPS is shown in Figure 4. The components of the EPS are a solar array (SA in Figure 4), a rechargeable nickel-cadmium battery (BA), a load representing all the electrical loads on board (LD), a relay (K1), and a device called a charge current controller (CCC) for controlling the relay. The solar array generates electricity when the satellite is in the sun, supplying power to the load and recharging the battery. The battery is a constant voltage source when it is charged between 6 and 30 ampere-hours. When the charge level is below 6 or above 30 ampere-hours, the electromagnetic force produced increases or decreases as it is charged or discharged.



SA: Solar array
LD: Electrical load on board
BA: Rechargeable battery
CCC: Charge current controller
t_1 through t_9 : Electrical terminals
s_1, s_2: Signal terminals
K1 : Relay
—— Signal connection
- - - - - Sensor data connection
—— Electrical connection

Figure 4: Electrical Power System

Since the battery can be damaged when it is charged beyond its capacity, the charge current controller opens the relays when the voltage reaches 33.8 volts to prevent the battery from being over-charged. The charge current controller (CCC) has a sensor connected to the the positive terminal (t_7) of the battery to sense the voltage. When it reaches 33.8 volts during a sun-light period, it turns on the relay K1. When the relay is energized, it opens and breaks the electrical connection, preventing further charging of the battery and switching the current source for the load from the solar array to the

battery. When the relay is open or when an eclipse period begins, the charge-level starts to decrease. When the charge-level decreases to 6.0, the voltage will start to decrease. At 31.0 volts the CCC turns K1 off to close if it has been opened.

The main purpose of the EPS is to supply electricity to the load constantly. This function of the EPS was shown in Figure 3, and the CPD's for achieving the function were shown in Figures 1 and 2.

2.1 EPS MODEL FRAGMENTS

The structure of EPS as shown in Figure 4 is given to DME as a collection of model fragments each representing a components and their connections. These model fragments represent the static aspect of the situation, and they are always active. In addition, there are model fragments representing various behavioral aspects of the components. They are activated/deactivated during the simulation according to the state of the world. Some of them are shown below with their conditions (P_m), behavior constraints (E_m), and results (R_m). Voltage and current are measured at terminals. The sign convention for current is that the current at a terminal is positive into the component owning the terminal. For the rest of the example, we will use the abbreviations shown below in parentheses to refer to the model fragments.

Model fragments concerning behaviors of battery

Battery-normal-operating-range (BN)

P_m: (Rechargeable-battery \$b) ∩ 6.0 amp-hours < (Charge-level \$b) < 30.0 amp-hours

E_m: (EMF \$b) = 33.0 volts

Battery-over-charged (BO)

P_m: (Rechargeable-battery \$b) ∩ (Charge-level \$b) > 30.0 amp-hours

E_m: (EMF \$b) = M^+(Charge-level \$b)

Battery-under-charged (BU)

P_m: (Rechargeable-battery \$b) ∩ (Charge-level \$b) < 6.0 amp-hours

E_m: (EMF \$b) = M^+(Charge-level \$b)

Behaviors of the solar array

Solar-array-generating (SG)

P_m: (Sun \$s) ∩ (Shining \$s) ∩ (Solar-array \$a) ∩ (In-closed-circuit \$a)

E_m: (Current-thru-terminal (Plus-terminal \$a)) < 0

Solar-array-in-eclipse (SE)

P_m: (Sun \$s) ∩ ~(Shining \$s) ∩ (Solar-array \$a)

E_m: (Current-thru-terminal (Plus-terminal \$a)) = 0

Solar-array-in-open-circuit (SO)
P_m: (Sun $s) ∩ (Shining $s) ∩ (Solar-array $a) ∩ ~(In-closed-circuit $a)
E_m: (Current-thru-terminal (Plus-terminal $a)) = 0

Behaviors of the charge current controller
Turn-k1-on (ON)
P_m: (Charge-current-controller $ccc) ∩ (Signal (Signal-terminal-1 $ccc)) = off ∩ (Voltage-at-terminal (Voltage-sensing-terminal $ccc)) ≥ 33.8 volts
R_m: (Signal (Signal-terminal-1 $ccc)) = on
Turn-k1-off (OFF)
P_m: (Charge-current-controller $ccc) ∩ (Signal (Signal-terminal-1 $ccc)) = on ∩ (Voltage-at-terminal (Voltage-sensing-terminal $ccc)) ≤ 31.0 volts
R_m: (Signal (Signal-terminal-1 $ccc)) = off

Behaviors of the relay
Relay-closed (RC)
P_m: (Relay $r) ∩ (Relay-closed-p $r)
E_m: (voltage-at-terminal (electrical-terminal-one $r)) = (voltage-at-terminal (electrical-terminal-two $r))
(current-thru-terminal (electrical-terminal-one $r)) = - (current-thru-terminal (electrical-terminal-two $r))
Relay-open (RO)
P_m: (Relay $r) ∩ ~ (Relay-closed-p $r)
E_m: (current-thru-terminal (electrical-terminal-one $r)) = - (current-thru-terminal (electrical-terminal-two $r))
Relay-closing (CL)
P_m: (Relay $r) ∩ ~(Relay-closed-p $r) ∩ (Signal (Signal-terminal $r) = off
R_m: (Relay-closed-p $r)
Relay-opening (OP)
P_m: (Relay $r) ∩ (Relay-closed-p $r) ∩ (Signal (Signal-terminal $r) = on
R_m: ~(Relay-closed-p $r)

In addition, there are three model fragments used to model the sun, Sun-rise (RISE), Sun-set (ST) and Reset-orbit-time (RST). As it takes approximately 100 minutes for the satellite to go around the earth once, Orbit-time is a 100-minute clock, which is reset to 0 when it reaches 100. We model the sun as rising and setting when Orbit-time = 0 and 60 respectively instead of modeling the satellite as revolving around the earth.

We will use the following notations for quantities.

I_i	Current through terminal t_i into the component owning t_i
V_i	Voltage measured at terminal t_i
C	The charge level of the battery
R_{ld}	The resistance of the load
R_{ba}	The internal resistance of the battery
EMF	The electromotive force of the battery
$Time$	Orbit-time

Each node in the behaviors of EPS can now be defined precisely in terms of these model fragments and their attributes. Using this vocabulary, the function of EPS shown in Figure 3 translates to the following condition:

EPS Function: $I_5[s] > 0$ for all $s \in Tr$.

The precise definition of each node in CPD_1 and CPD_2 using these model fragments and their attributes are shown in parentheses in Figures 1 and 2.

3.1 SIMULATED BEHAVIOR.

We simulated the behavior of EPS on DME. In the initial state, *Time* is between 0 and 60, sun is up, the relay is closed, and the charge level is between 6 and 30.0 amp-hours. From this initial state, a number of behaviors are possible. In qualitative simulation mode, because of the ambiguity of qualitative simulation, there are multiple possible trajectories. Tables 1 presents one of the possible trajectories of EPS generated by DME. The variable values are shown with their magnitude and the sign of their derivative. In a state where the derivative is undefined, the sign is shown as x. The right most column of the table shows the set of active models in each state. The set of all active model fragments in each state is actually much larger, but since most of them represent components, terminals, junctions, etc. and are active throughout the simulation, we show only the ones that change their activation status at some point. A model fragment that becomes activated in a given state is shown in bold. A × over a model fragment indicates that it becomes deactivated in the given state.

3.1.1 *Trajectory* Tr_1. The behavior we consider is summarized in Table 1. In the initial state s_0, since the sun is up and the relays are closed, the charge-level is increasing. When it eventually reaches 30 amp-hours (s_1), the battery enters the over-charged state and the voltage level starts to rise. When it reaches 33.8 (s_3), CCC changes the signal to on (s_4), and K1 opens (s_5). At this point, the solar array stops generating current, and the battery starts to discharge. The charge level and the voltage starts to decrease. Soon, the sun sets (s_7). As the charge level continues to decrease, the battery returns to the normal operating range (s_9) It eventually becomes under-charged (s_{11}), and

the voltage starts to decrease below 33.0 volts. When it reaches 31.0 volts ($s12$), the signal to K1 is turned off ($s13$), and K1 closes ($s14$). Soon, the sun rises again ($s17$), and charging resumes.

Following are equations that are generated by DME during the simulation. On the right of each equation, we indicate the model fragment that gives rise to the equation. "Junction" indicates that the equation was generated as a behavior constraint of electrical junction model fragments (not shown in Figure 4 to simplify the figure.) Note that $e5'$ is applicable when $e5$ is not, and vice versa.

$e1: I2 = -I1$	*SG*	$e7: V5 = R_{ld}\, I5$	*LD*
$e2: I2 + I3 = 0$	*junction*	$e8: V7 = V5$	*junction*
$e3: I3 + I4 = 0$	*RC*	$e9: V5 = V4$	*junction*
$e4: I4 + I5 + I7 = 0$	*junction*	$e10: V4 = V3$	*RC*
$e5: EMF = M+(C)$	*BO or BU*	$e11: V3 = V2$	*junction*
$e5': EMF = 33.0$	*BN*	$e12: dC/dt = I7$	*BA*
$e6: V7 = EMF + R_{ba}\, I7$	*BA*		

R_{ld} and R_{ba} are exogenous.

The causal ordering among variables in the states where the relay is closed is shown in Figure 5. The causal ordering when the relay is open is mostly similar except that the links from $I2$ to $I3$ and from $I3$ to $I4$ are missing[1]. In the figure, the variable at the head of an arrow is causally dependent on the variable at the tail. The causal links are labeled with equations that are responsible for the link. The link labeled i is an integration link to a variable from its derivative. Variables $I5$, $V5$, $I7$, and $V7$ are inter-dependent. The equations responsible for their inter-dependence are $e4$, $e6$, $e7$ and $e8$.

We prove that this trajectory satisfies the expected behavior of EPS. Due to the limitation of space, we omit the proof that the qualifiers, *Provided*, and *By-function-of*, on causal links are satisfied. The proof of these conditions is straightforward.

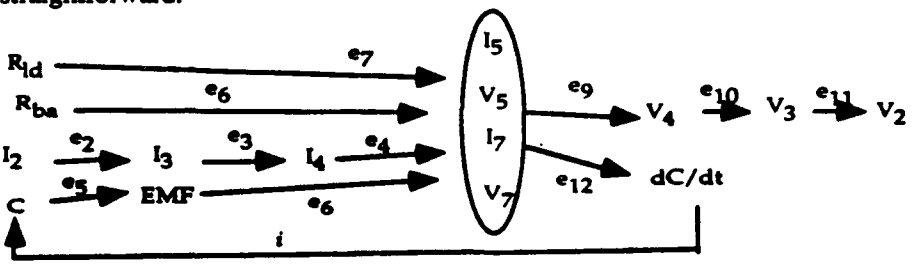


Figure 5: Causal ordering when K1 is closed

1 When the battery is in its normal operating range, the causal ordering is slightly different since $e5'$ instead of $e5$ is applicable, eliminating the link from *C* to *EMF*.

Table 1: Trajectory, Tr_1, of EPS

state	Sun	Relay	Signal	I_2	I_5	I_7	V_7	c	Time	Active Models
s_0	shine	close	off	- std	+ std	+ std	33.0 std	6-30 inc	0-60 inc	BN, SG, RC
s_1								30 inc		BN, SG, RC
s_2							33.0-33.8 inc	30-inf inc		BO, SG, RC
s_3							33.8 x			BO, SG, RC, ON
s_4			on							BO, SG, RC, ON, OP
s_5		open		0 stc		- std	33.0-33.8 dec	30-inf dec		BO, ST, SO, RO
s_6									60 inc	BO, ST, SO, RO
s_7	-shine								60-100 inc	BO, ST, SE, SO, RO
s_8							33.0 dec	30 dec		BO, SE, RO
s_9							33.0 std	6-30 dec		BN, SE, RO
s_{10}								6 dec		BN, SE, RO
s_{11}							31.0-33.0 dec	0-6 dec		BU, SE, RO
s_{12}							31.0 dec			BU, SE, RO, OFF,
s_{13}			off				0-31.0 dec			BU, SE, RO, OFF, CL
s_{14}		close								BU, SE, RST, CL, RC
s_{15}									100 x	BU, SE, RST, RC
s_{16}									0 x	BU, SE, RST, RIBE, RC
s_{17}	shine			- std		+ std	0-31.0 inc	0-6 inc	0 inc	BU, SE, RST, SG, RC
s_{18}									0-60 inc	BU, SG, RC
s_{19}							31.0 inc			BU, SG, RC
s_{20}							31.0-33.0 inc			BU, SG, RC
s_{21}							33.0 inc	6 inc		BU, SG, RC
s_{22}							33.0 std	6-30 inc		BN, SG, RC

(1) That the function of EPS is satisfied by behavior Tr_1 is clear from Table 1 since $I_5 > 0$ in all states. That EPS or its components takes part in bringing about the fulfillment of the goal is subsumed by the proof in part (2) and (3).

(2) The following proof applies for s being one of states s_0 to s_4, and s_{17} through s_{22}, where the condition of CPD_1, *(Shining Sun)*, holds.

Let $st(n_1) = st(n_2) = st(n_3) = st(n_4) = s$. It follows that $st(n_1) \leq st(n_2) \leq st(s_3) \leq st(s_4)$. Since *(Shining Sun)*$[s] \cap (< I_2\ 0)[s] \cap (> dC/dt\ 0)[s] \cap (> I_5\ 0)[s]$, we have $def(n)[s]$ for all n in CPD_1.

Proof of l_1: *(Shining Sun)*$[s] \Rightarrow_C I_2[s]$.

(Shining Sun)$[s] \Rightarrow_C SG[s]$ because of Definition 5.c.

$SG[s] \Rightarrow_C I_2[s]$ because of Definition 5.e.

It follows that *(Shining Sun)*$[s] \Rightarrow_C I_2[s]$.

Proof of l_2: $I_2[s] \Rightarrow_C dC/dt[s]$

$I_2 \rightarrow_C dC/dt$ in s as shown in Figure 5.

It follows that $I_2[s] \Rightarrow_C dC/dt[s]$ because of Definition 5.d.

Proof of l_3: $I_2[s] \Rightarrow_C I_5[s]$

$I_2 \rightarrow_C I_5$ in s as shown in Figure 5.

It follows that $I_2[s] \Rightarrow_C I_5$ because of Definition 5.d.

Therefore, CPD_1 of EPS is achieved in states s_0, s_1, and s_{17} through s_{23} of Tr_1.

(3) The following is true for s being one of states s_5 through s_{16}, where the condition of CPD_2, *~(Shining Sun)* $\cup$ *(Active BO)*, holds.

Let $st(n_5) = st(n_6) = s$. It follows that $st(n_5) \leq st(n_6)$.

Since $(< dC/dt\ 0)[s] \cap (> I_5\ 0)[s]$, we have $def(n)[s]$ for all nodes n in CPD_2.

Proof of l_4: $I_5[s] \Rightarrow_C dC/dt[s] \cap dC/dt[s] \Rightarrow_C I_5[s]$

$I_5 \rightarrow_C dC/dt$ in s as shown in Figure 5. It follows that $I_5[s] \Rightarrow_C dC/dt[s]$ because of Definition 5.d.

Therefore, CPD_2 of EPS is achieved in states s_2 through s_{16} of Tr_1.

3. Discussion

In this paper, we formalized a number of notions that were relatively informally specified in the Functional Representation language and defined matching between an expected behavior represented in Functional Representation and a predicted behavior. We also demonstrated its use in deciding whether a particular trajectory of a device achieves an expected behavior.

Our primary goal is to use the knowledge of functions and expected behavior for the purpose of design verification. It is important that the definition of behavior verification we have presented is not biased towards any particular perspective about what are more important than others as a causal factor. In other words, it does not require that the function or the expected behavior be described from a particular point of view. This definition of verification of a predicted behavior with respect to a function and an expected behavior is inclusive enough to allow a trajectory to match many representations of functions or expected behaviors. Likewise, there can be any number of trajectories that can be shown to match a given expected behavior as there can be any number of designs that accomplish the same functionality. Thus, the mapping between trajectories and an expected behavior is many to many. However, if the goal is to verify that a predicted behavior achieves a given expected behavior, this non-uniqueness of a match is not a problem. Our definition does not establish that the given expected behavior is the only correct causal story for a given trajectory, nor that the trajectory is the only correct way to achieve the function. However, the definition does establish that a given design achieves the function in an expected manner, which is what is needed for our purpose of design verification.

Our next step is to implement a program that takes a functional representation and a trajectory and automatically proves whether or not the expected behavior is realized in the trajectory.

3.1 RELATED WORK

Bradshaw and Young (1991) and Franke (1991) have also proposed representations of the knowledge of a purpose and their use in design. They represent the intended function in a manner that is similar to the way functions are represented in Functional Representation. Bradshaw and Young built a system called Doris, which uses knowledge of purpose for evaluating behaviors generated by qualitative simulation as well as for diagnosis and explanation.

The focus of Franke's work on representing functions is slightly different from ours or Bradshaw and Young in that he represents the purpose of a design modification and not that of a whole device. He developed a representation scheme, called TED, in which he expresses the purpose for making a modification δ in a structure using the same function types as those in Functional Representation. Thus, in order to prove that a function is achieved by a modification δ, he must compare the behavior of structure M and that of M', which is M with the modification δ. Another important characteristic of TED's representation of functions is that it can be a

sequence (not necessarily a linear) of partial descriptions. The representation of a function in TED typically says "δ guarantees σ," where σ is a sequence, called scenario, of partial descriptions. The sequence of partial descriptions is matched against states in a sequence of qualitative states generated by QSIM.

The most important difference between our work and the works by Franke's or by Bradshaw and Young's is that we take not only the functions but also the causal interactions into account in evaluating behavior. We feel that it is important to test whether it is in fact the causal processes intended by the designer that are responsible for bringing about the achievement of the functional goal, since the satisfaction of the functional goal does not necessarily indicate that the design is functioning as intended. We believe that evaluating a trajectory with respect to the causal process as well as the function allows one to uncover hidden flaws in a design which may otherwise go undetected.

References

Bradshaw, J. A. and Young, R. M.: 1991, Evaluating Design Using Knowledge of Purpose and Knowledge of Structure. *IEEE Expert*, April.

Crawford, J., Farquhar, A., and Kuipers B.: 1990, QPC: A Compiler from Physical Models into Qualitative Differential Equations. *Proceedings of the Eighth National Conference on Artificial Intelligence.*

Falkenhainer, B. and Forbus, K.: 1988, Setting up Large-Scale Qualitative Models. *Proceedings of the Seventh National Conference on Artificial Intelligence.*

Fikes, R., Gruber, T., Iwasaki, Y., Levy, A. and Nayak, P.: 1991, How Things Work Project Overview. Technical Report, KSL 91-70, Knowledge Systems Laboratory, Stanford University.

Franke, D. W.: 1991, Deriving and Using Descriptions of Purpose. *IEEE Expert*, April.

Iwasaki, Y. and Low, C. M.: 1991, Model Generation and Simulation of Device Behavior with Continuous and Discrete Changes. Technical Report KSL-91-69, Knowledge Systems Laboratory, Stanford University.

Iwasaki, Y. and Simon, H.A.:1986, Causality in Device Behavior. *Artificial Intelligence* 29.

Keuneke, A.: 1991, Device Representation: The Significance of Functional Knowledge. *IEEE Expert*, April.

Kuipers, B.: 1986, Qualitative Simulation. *Artificial Intelligence* 29.

LMSC: 1984, *Support Systems Module System Procedure for Pointing and Control Subsystem (SE-23, Vol. V)*, Lockheed Missiles and Space Company document # D889545A.

Sembugamoorthy, V. and Chandrasekaran, B.: 1986, Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems, *in* Kolodner, J.L. and Riesbeck, C.K. (eds), *Experience, Memory, and Reasoning*, Lawrence Erlbaum Associates, Hillsdale, NJ.

AAAI 95

CFRL: A Language for Specifying the Causal Functionality of Engineered Devices

Marcos Vescovi Yumi Iwasaki Richard Fikes B. Chandrasekaran

Knowledge Systems Laboratory
Stanford University
701 Welch Road, Bldg C
Palo Alto, CA 94304
vescovi,iwasaki,fikes@ksl.stanford.edu

Laboratory for AI Research
The Ohio State University
217 B, Bolz Hall, 2036 Neil Avenue
Columbus, OH 43210-1277
chandra@cis.ohio-state.edu

Abstract[1]

Understanding the design of an engineered device requires both knowledge of the general physical principles that determine the behavior of the device and knowledge of what the device is intended to do (i.e., its functional specification). However, the majority of work in model-based reasoning about device behavior has focused on modeling a device in terms of general physical principles or intended functionality, but not both. In order to use both functional and behavioral knowledge in understanding a device design, it is crucial that the functional knowledge is represented in such a way that it has a clear interpretation in terms of actual behavior. We propose a new formalism for representing device functions with well-defined semantics in terms of actual behavior. We call the language CFRL (Causal Functional Representation Language). CFRL allows the specification of conditions that a behavior must satisfy, such as occurrence of a temporal sequence of expected events and causal relations among the events and the behavior of device components. We have used CFRL as the basis for a functional verification program which determines whether a behavior achieves an intended function.

Introduction

Understanding the design of an engineered device requires both knowledge of the general physical principles that determine the behavior of the device and knowledge of what the device is intended to do (i.e., its functional specification). However, the majority of work in model-based reasoning about device behavior has focused on modeling a device in terms of general physical principles *or* intended functionality, but not both. For example, most of the work in qualitative physics has been concerned with predicting the behavior of a device given its physical structure and knowledge of general physical principles. In that work, great importance has been placed on preventing a pre-conceived notion of an intended function of the device from influencing the system's reasoning methods and representation of physical principles in order to guarantee a high level of "objective truth" in the predicted behavior. In contrast, in their work based on the FR (Functional Representation) language (Sembugamoorthy & Chandrasekaran 1986) (Keuneke 1986), Chandrasekaran and his colleagues have focused mostly on modeling a device in terms of what the device is intended to do and how those intentions are to be accomplished through causal interactions among components of the device.

Both types of knowledge, functional and behavioral, seem to be indispensable in fully understanding a device design. On the one hand, knowledge of intended function alone does not enable one to reason about what a device might do when it is placed in an unexpected condition or to infer the behavior of an unfamiliar device from its structure. On the other hand, knowledge of device structure and general physical principles may allow one to predict how the device will behave under a given condition, but without knowledge of the intended functions, it is impossible to determine if the predicted behavior is a desirable one, or what aspect of the behavior is significant.

In order to use both functional and behavioral knowledge in understanding a device design, it is crucial that the functional knowledge is represented in such a way that it has a clear interpretation in terms of actual behavior. Suppose, for example, that the function of a charge current controller is to prevent damage to a battery by cutting off the charge current when the battery is fully charged. To be able to determine whether this function is actually accomplished by an observed behavior of the device, the representation of the function must specify conditions that can be evaluated against the behavior. Such conditions might include occurrence of a temporal sequence of expected events and causal relations among the events and the components. Without a clear semantics given to a representation of functions in terms of actual behavior, it would be impossible to evaluate a design based on its predicted behavior and intended functions.

While it is important for a functional specification to have a clear interpretation in terms of actual behavior, it is also desirable for the language for specifying functions to be independent of any particular system used for simulation. Though there are a number of alternative methods for predicting behavior, such as numerical simulation with discrete time steps or qualitative

[1]The research by the first three authors is supported in part by the Advanced Research Projects Agency, ARPA Order 8607, monitored by NASA Ames Research Center under grant NAG 2-581, and by NASA Ames Research Center under grant NCC 2-537. Chandrasekaran's research is supported by the Advanced Research Projects Agency by means of AFOSR contract F-49620-89-C-0110 and AFOSR grant 89-0250.

simulation, a functional specification at some abstract level should be intuitively understandable without specifying a particular simulation mechanism. If a functional specification language was dependent on a specific simulation language or mechanism, a separate functional specification language would be needed for each different simulation language, which is clearly undesirable. What is needed is a functional specification language that has sufficient expressive power to support descriptions of the desired functions of a variety of devices. At the same time, the language should be clear enough so that for each simulation mechanism used, it can be given an unambiguous interpretation in terms of a simulated behavior.

An essential element in the description of a function is causality. In order to say that a device has achieved a function, which may be expressed as a condition on the state of the world, one must show not only that the condition is satisfied but also that the device has participated in the causal process that has brought about the condition. For example, when an engineer designs a thermostat to keep room temperature constant, the design embodies her idea about how the device is to work. In fact, the essential part of her knowledge of its function is the expected causal chain of events in which it will take part in achieving the goal. Thus, a representation formalism of functions must provide a means of expressing knowledge about such causal processes.

We have developed a new representational formalism for representing device functions called CFRL (Causal Functional Representation Language) that allows functions to be expressed in terms of expected causal chains of events. We have also provided the language with a well-defined semantics in terms of the type of behavior representation widely used in model-based, qualitative simulation. Finally, we have used CFRL as the basis for a functional verification program which determines whether a behavior achieves an intended function.

This paper is organized as follows: We first describe the representation of behavior over time in terms of which the semantics of CFRL will be defined and our assumptions about the modeling and simulation schemes that produce such a behavior description. We then present the CFRL language and define its semantics in terms of behavior. We close with a discussion and summary.

Behavior Representation

Before describing CFRL, we briefly describe the behavior representation in terms of which the semantics of CFRL will be defined. A physical situation is modeled as a collection of *model fragments*, each of which represents a physical object or a conceptually distinct physical phenomenon, such as a particular aspect of component behavior or a physical process. A model fragment representing a phenomenon specifies a set of conditions under which the phenomenon occurs and a set of consequences of the phenomenon. The conditions specify a set of instances of object classes that must exist and a set of relations that must hold among those objects and their attributes for the phenomenon to occur. The consequences specify the functional relations the phenomenon will cause to hold among the objects and their attributes.

Model fragments can represent phenomena as occurring continuously while the fragment's conditions hold or as events that occur instantaneously when the conditions become true. The consequences of a model fragment that represents an event are facts to be asserted resulting from the event, whereas the consequences of a model fragment that represents a continuous process are sentences (e.g., ordinary differential equations) which are true while the phenomena is occurring.

When there exists at time t a set of objects represented by model fragments m_i to m_j that satisfy the conditions of a model fragment m_0, we say that an instance of m_0 is active at that time. We will call m_i through m_j the *participants* of the m_0 instance.

Representation of physical knowledge in terms of model fragments is a generalization of the representation of physical processes and individuals in Qualitative Process Theory (Forbus 1984). There are several systems, including the Device Modeling Environment (DME) (Iwasaki & Low 1991) the Qualitative Process Engine (QPE) (Forbus 1989), and the Qualitative Process Compiler (QPC) (Crawford, Farquhar & Kuipers), that use similar representations for physical knowledge to predict the behavior of physical devices over time. Though the ways these systems actually perform prediction differ, the basic idea behind all of them is the following: For a given situation, the system identifies active model fragment instances by evaluating their conditions. The active instances give rise to equations representing the functional relations that must hold among variables as a consequence of the phenomena taking place. The equations are then used to determine the next state into which the device must move.

We assume that a behavior is a linear sequence of states. The output of a qualitative simulation system such as QPE, DME, and QPC is usually a tree or a graph of states. Each path through the graph represents a possible behavior over time. We will refer to such a path, i.e., a linear sequence of states, as a *trajectory*.

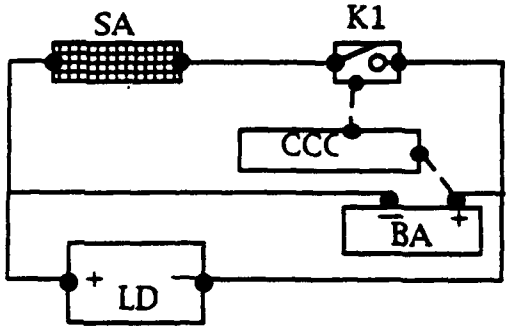
A state represents a situation in which the physical system being modeled is in at a particular time. "A particular time" here can be a time point or interval. We will not assume any specific model of time in this paper. The only assumptions about time that we make are: (1) the times associated with different states do not overlap; (2) when a state s_j immediately follows s_i in a behavior, there is no other "time" that falls between the times (periods) associated with s_i and s_j; and (3) every state has a unique successor (predecessor) unless it is the final (initial) state, in which case it has none.

In our modeling scheme, each state has a set of variable values and predicates that hold in the state. In addition,

each state has a set of active model fragment instances representing the phenomena that are occurring in the state.

An Electrical Power System

This section presents the device that we will use throughout the rest of this paper as an example. The device is the electrical power system (EPS) aboard an Earth orbiting satellite (Lockheed 1984). A simplified schematic diagram of the EPS is shown in Figure 1. The main purpose of the EPS is to supply a constant source of electricity to the satellite's other subsystems. The solar array generates electricity when the satellite is in the sun, supplying power to the load and recharging the battery. The battery is a constant voltage source when it is charged between 6 and 30 ampere-hours. When the charge level is below 6 ampere-hours, the voltage output decreases as the battery discharges. When the charge level is above 30 ampere-hours, the voltage output increases as it is charged.



SA: Solar array
LD: Electrical load on board
BA: Rechargeable battery
CCC: Charge current controller
K1: Relay

Figure 1: An Electrical Power System.

Since the battery can be damaged when it is charged beyond its capacity, the charge current controller opens the relay when the voltage exceeds a threshold to prevent the battery from being over-charged. The controller senses the voltage via a sensor connected to the positive terminal of the battery. When the voltage is greater than 33.8 volts, the controller turns on the relay K1. When the relay is energized, it opens and breaks the electrical connection to prevent further charging of the battery, thereby switching the current source for the load from the solar array to the battery. When the relay is open or when an eclipse period begins, the battery's charge-level starts to decrease. When the battery becomes under-charged, the voltage decreases. When it reaches 31.0 volts, the CCC turns relay K1 off to close it.

CFRL

We now describe the syntax and semantics of CFRL. Figures 2 shows an example of the representation of a function of the EPS.

```
DF:  ?eps: Electrical-power-system
CF:  Object-set: ?sun: Sun    ?l: electrical-load
     Conditions: T
GF:
 (ALWAYS
   (AND
     (-> (AND (Shining-p ?sun)
              (Closed-p (Relay-component ?eps)))
         CPD1)
     (-> (OR (NOT (Shining-p ?sun))
             (Open-p (Relay-component ?eps)))
         CPD2)
     (-> (AND  (> (Electromotive-force
                    (Battery-component ?eps))
                  33.8)
               (Closed-p (Relay-component ?eps)))
         CPD3)
     (-> (AND (< (Electromotive-force
                   (Battery-component ?eps))
                 31.0)
              (Open-p (Relay-component ?eps)))
         CPD4)))
```

Figure 2-a: Function F_1 of EPS

We consider a function to be an agent's belief about how an *object* is to be used in some *context* to achieve some *effect*. Thus, our representation of a function specifies the object, the context, and the effect. However, it does not specify an agent, which is implicitly assumed to be whoever is using the representation. Formally, a function is defined as follows:

Definition 1: Function

A function F is a triplet $\{D_F, C_F, G_F\}$, where:

D_F denotes the device of which F is a function.

C_F denotes the context in which the device is to function.

G_F denotes the functional goal to be achieved.

The device specification, D_F, specifies the class of the device and the symbol by which the device will be referred to in the rest of the definition of F. The example in Figure 2-a states that the function is of an Electrical-power-system which will be referred to as *?eps* in the rest of the definition.

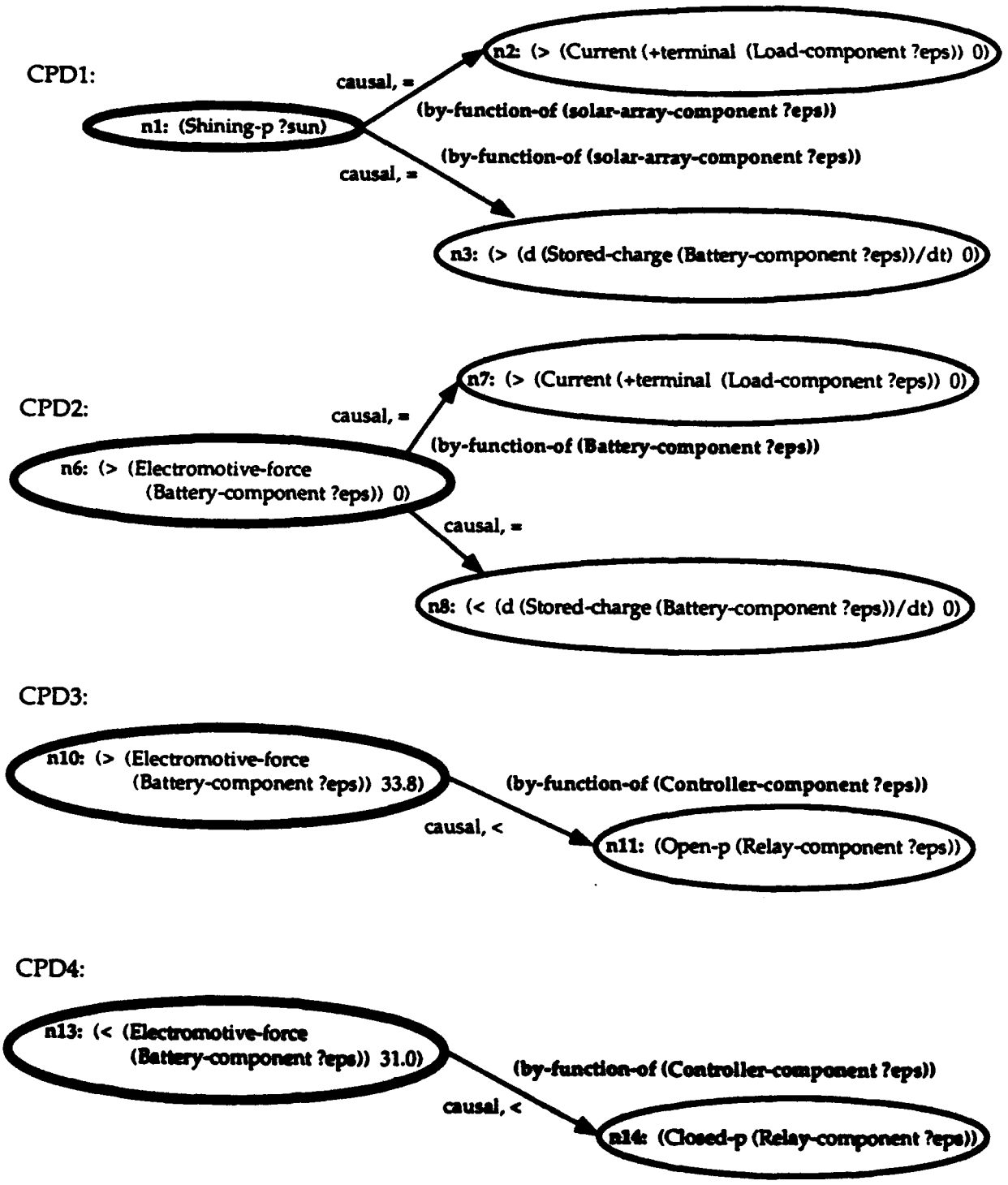


Figure 2-b: CPD's of Function F_1 of EPS.

The notion of a device function assumes some physical context in which the device is placed, and C_F is a specification of such a context. C_F consists of two parts, a set of objects and a set of conditions on those objects. For example, Figure 2-a states that there must exist an instance of Sun and an instance of electrical load. The conditions must hold throughout a behavior in order for the function to be verified in the behavior.

Formally, the *Object-set* of a C_F is a list of pairs *(var. type)*, where *var* is a symbol to be used in the description of F to refer to the object, and *type* is the type (class) of the object. *Conditions* is a logical expression involving the variables defined in the *Object-set* and D_F.

The third part of the function definition, G_F, specifies the behavior to be achieved by the device used in a specific manner. G_F of a function is represented as a Boolean combination of *Causal Process Descriptions* (CPDs) and conditions involving the variables defined in D_F and the *Object-set* of C_F. Each CPD is an abstract description of expected behavior in terms of a causal sequence of events. In the following, we formally define a CPD.

Causal Process Descriptions (CPD's)

Figure 2-b shows examples of CPD's which are part of the functional specification of the EPS. A CPD is a directed graph, in which each node describes a state and each arc describes a temporal and (optionally) a causal relation between states.

A node specifies a condition on a state. The condition is a logical sentence about the state of the world at some time using the variables defined in the D_F and C_F portions of the function. For example, the node n_1 in Figure 2-b states the condition that the sun be shining. One or more nodes in each CPD are distinguished as the initial node(s). In the figures, the initial nodes are indicated with a thick oval. A condition specified by a node can contain AND and OR as logical connectives. When the meaning is clear, we will use the name of a node to refer to the condition represented by the node.

The arcs in a CPD are directed and specify temporal and causal relations among nodes. An arc has the following attributes:

source: The node at the tail of the arc.

destination: The node at the head of the arc.

causal-flag: An indicator of whether the relationship between the states described by the source and destination nodes is causal. (The relationship is always temporal.)

temporal-relation: $=$, $<$, or $\leq$, indicating the temporal relation between the states described by the source and destination nodes. $=$ means that the states described by the two nodes are to be the same state, $<$ means the state described by the source node must strictly precede the state described by the destination node, and $\leq$ means the state described by the source node must either be the same as or precede the state described by the destination state.

causal-justification: If an arc is "causal", one can attach a justification for the causal relation. A justification takes the form of a Boolean combination of the following predicates:

(by-function-of <model-fragment>),
(with-participation-of <model-fragment>).

The meaning of these predicates will be explained after we give a precise definition of a *causal relation* among nodes.

In order to refer to attributes of arcs, we will use the attribute name (e.g., source, destination, etc.) as a function of the arc as in "*source(a1)*".

We will write $n_i \Rightarrow_c n_j$ when there is a causal arc from n_i to n_j. As a condition specified by a node can be a Boolean combination of conditions, the following defines the meaning of causal relations among them, where $e1$, $e2$, and $e3$ are conditions:

a) $(\text{AND } e1\ e2) \Rightarrow_c e3 \equiv (\text{AND } (e1 \Rightarrow_c e3)\ (e2 \Rightarrow_c e3))$

b) $e1 \Rightarrow_c (\text{AND } e2\ e3) \equiv (\text{AND } (e1 \Rightarrow_c e2)(e1 \Rightarrow_c e3))$

c) $(\text{OR } e1\ e2) \Rightarrow_c e3 \equiv (\text{OR } (e1 \Rightarrow_c e3)\ (e2 \Rightarrow_c e3))$

d) $e1 \Rightarrow_c (\text{OR } e2\ e3) \equiv (\text{OR } (e1 \Rightarrow_c e2)\ (e1 \Rightarrow_c e3))$

Semantics of a CPD

A CPD can be considered to be an abstract specification of a behavior. Unlike a trajectory, it does not specify every state or everything known about each state. It only specifies some of the facts that should be true during the course of the behavior and partial temporal/causal orderings among those facts. The intuitive meaning of a CPD is that:

- For each node in the CPD, there must be a state in the trajectory in which the condition specified by the node is satisfied, and
- For each pair of nodes directly connected by an arc, the causal and temporal relationships specified by the arc must exist in the trajectory.

In order for us to evaluate these conditions against a behavior, we must define their meanings in terms of the languages used to describe a (simulated or actual) behavior. In this paper, we will do so in terms of the behavior representation formalism described earlier.

However, note that CFRL itself is independent of the particular behavior representation language used, and that one would need to provide different definitions in order to evaluate functional specifications in CFRL against behaviors generated by a different scheme.

We first present the definition of a *causal dependency* relation between sentences in a trajectory and the *causality constraints* that can be associated with a CPD arc. We then define the requirements for a trajectory to match a CPD and for a trajectory to match a function goal. Finally, we use those definitions to define the requirements for a trajectory to achieve a function.

A few words about notation: We will attach $[s]$ to a sentence to denote the sentence holds in state s. Therefore, $p[s]$ means that p holds in state s. We will also associate a state with models and variables to denote sentences as follows:

$m[s]$: An instance of model fragment m is active in s.

$v[s]$: The value of variable v in s. (i.e., an axiom of the form $(= (value\ v\ s)\ c)$ for some constant c.)

We will use the relations $<$, $>$, $=$, and $\leq$ to express temporal ordering among states in a trajectory. For example, for states $s1$ and $s2$ in a trajectory, "$s1 < s2$" means that $s1$ strictly precedes $s2$ in time. Note that ordering is total for states in a trajectory because a trajectory is a linear sequence of states, while the ordering is partial for states in a CPD.

Intuitively, we say $p2$ is *causally dependent* on $p1$ in trajectory Tr, written $p1 \Rightarrow p2$, when it can be shown that $p1$ being true in Tr eventually leads to $p2$ being true in Tr.

Definition 2: Causal Dependency

The causal dependency relation, $\Rightarrow$, is a binary relation between sentences in a trajectory with the following properties:

1. For all atomic sentences p, states s, model fragments m, and variables v:

   a) If $p[s0], p[s1], \ldots p[s]$ (i.e., if p is part of the initial conditions and is never changed), then $\phi \Rightarrow p[s]$. (And we say that $p[s]$ is exogenous.)

   b) If model fragment m represents an event and asserts p, and if there exists a state s_j such that $s_j < s$, $\neg p[s_j]$, $m[s_j]$, and $p[s_k]$ for all $k > j$ (i.e., p became true at some point before s due to m), then $m[s_j] \Rightarrow p[s]$.

   c) If model fragment m represents a continuous process and has p as a consequence, and if there exists a state s_j such that $s_j < s$, $\neg p[s_j]$, $m[s_j]$, and $p[s_k]$ for all $k > j$ (i.e., p became true at some point before s due to m), then $m[s_j] \Rightarrow p[s]$.

   d) If model fragment m has p as a condition, then $p[s] \Rightarrow m[s]$.

   e) If v occurs in p as a term and p is not $v[s]$, then $v[s] \Rightarrow p[s]$.

   f) If v is an exogenous variable, $\phi \Rightarrow v[s]$.

   g) For all variables v' such that $v' \rightarrow v$ is in the causal ordering[2] in s:

      (i) $v'[s] \Rightarrow v[s]$;

      (ii) If $p[s]$ is the equation through which v depends on v', then $p[s] \Rightarrow v[s]$.

   h) For all variables v' such that v and v' are in a feedback loop in the causal ordering in s:

      (i) $v'[s] \Rightarrow v[s]$ and $v[s] \Rightarrow v'[s]$;

      (ii) For each equation p such that p is part of the feedback loop and v appears in p, $p[s] \Rightarrow v[s]$.

   i) If $s1$ is the state immediately following s, and dv is the time-derivative of v in s, then $dv[s] \Rightarrow v[s1]$.

2. $\Rightarrow$ is transitive.

When $p_i \Rightarrow p_j$, we will say that p_j is *causally dependent on* p_i or that p_i *causes* p_j. Given statements $p[s_i]$ and $p[s_j]$ such that $p[s_i] \Rightarrow p[s_j]$, we call the causal sequence of statements starting from $p[s_i]$ and leading to $p[s_j]$ the *causal path* from $p[s_i]$ to $p[s_j]$.

Having defined the meaning of a causal relation among statements, we can now explain the meaning of the predicates used to justify causal arcs in a CPD.

Definition 3: Causality constraints

Given an arc a from node n_i to n_j in a CPD and a model fragment m, causality constraints of the following form can be associated with a:

a) ***(by-function-of m)*** -- meaning that the causal path from n_i to n_j includes a consequence of an instance of m;

b) ***(with-participation-of m)*** -- meaning that the causal path from n_i to n_j includes a consequence of an instance of a model fragment in which an instance of m participates.

These predicates do not imply specific commitments as to *how* the components participate in the causal process. They give the designer the capability of using whatever component has the desired function, independent of its particular mechanism.

We can now present the definitions on which verification of a trajectory with respect to a CPD is based.

Definition 4: Matching of a state and a node

A state s in a trajectory and a node n in a CPD are said to *match* if the condition specified in n is true in s.

Having defined the meaning of a causal relation among statements in a trajectory, we can now define the meaning

[2] Causal ordering is a technique for determining causal dependency relations among variables in a set of equations (Iwasaki & Simon 1986).

of the causal and temporal relations between linked nodes of a CPD.

Definition 5: Satisfying the constraints of an arc

If a is an arc from node n_i to n_j in a CPD, then the causal and temporal constraints of a are satisfied at states s_i and s_j if both of the following conditions are satisfied:

a) $s_i < (= \text{or} \leq)\ s_j$ when $n_i < (= \text{or} \leq)\ n_j$, respectively.

b) If arc a is causal and if n_i and/or n_j are Boolean combinations of conditions, then the causal relation between n_i and n_j can be rewritten as a Boolean combination of causal relations of the form $e_i \Rightarrow_C e_j$, where e_i and e_j are atomic conditions. $e_i[s_i] \Rightarrow_C e_j[s_j]$ is satisfied if for every variable[3] v_i used in e_i and every variable v_j used in e_j, $v_i[s_i] \Rightarrow v_j[s_j]$ and the causal path from $v_i[s_i]$ to $v_j[s_j]$ satisfies the causal justification on a.

Definition 6: Matching of a CPD and a trajectory

Let T be a trajectory consisting of a linear sequence of m states, s_1 through s_m. Let CPD_1 be a CPD consisting of a set of nodes, N_1, and a set of arcs, A_1. CPD_1 and T are said to match iff all the following conditions are satisfied:

a) The initial nodes of CPD_1 match the initial state s_1 in T.

b) For each remaining node n in N_1, there exists a state in T that matches n such that for every arc a in A_1 from nodes n_i to n_j, the temporal and causal constraints specified by a are satisfied by the states matched to n_i and n_j.

Representation of the Functional Goal (G_F)

The functional goal of a function (denoted by G_F) is represented as an expression consisting of CPDs, conditions, quantifiers, and Boolean connectives. Nested expressions using connectives are allowed, but a quantifier cannot appear in the scope of another quantifier. Each CPD must appear in the scope of one and only one quantifier. There are two quantifiers, ALWAYS and SOMETIMES. Connectives are AND, OR, IMPLIES, and NOT. Syntactically, the connectives are used in the same way as ordinary logical connectives. The following are example G_F expressions:

(ALWAYS (AND *cpd1 cpd2* (OR *cpd3 cpd4*)))
(OR (ALWAYS *cpd1*)
    (SOMETIMES (AND *cpd2 cpd3*)))
(ALWAYS (NOT *cpd1*))

[3] The variables used in CFRL can be different from the variables in terms of which the trajectory states are defined, since CFRL descriptions represent a device-level perspective, while states in the trajectory represent a component or physical process-level perspective. Correspondences between CPD variables and trajectory variables are made when the function is matched against a specific trajectory.

Quantifiers align the initial nodes of the CPDs in their scope as well as specify whether the described behavior must hold in every subsequence of the trajectory or only in some of them. The connectives and quantifiers are to be interpreted as specified in the following definition of matching a G_F and a trajectory.

Definition 7: Matching of a G_F and a trajectory

Let T be a trajectory consisting of a linear sequence of m states, s_1 through s_m; T_i denote subsequences of T from s_i through s_m; and <cpd-exp> denote a Boolean combination of CPD's and conditions. Then:

a) *(ALWAYS <cpd-exp>)* matches T iff *<cpd-exp>* matches T_i for each T_i ($i = 1$ to m).

b) *(SOMETIMES <cpd-exp>)* matches T iff *<cpd-exp>* matches T_i for some T_i ($i = 1$ to m).

c) *(AND <cpd-exp0> <cpd-exp1> ...)* matches T iff every conjunct matches T.

d) *(OR <cpd-exp0> <cpd-exp1> ...)* matches T iff at least one of the disjuncts matches T.

e) *(NOT <cpd-exp>)* matches T iff *<cpd-exp>* does not match T.

f) *(IMPLIES <cpd-exp0> <cpd-exp1>)* matches T iff *<cpd-exp0>* does not match T or *<cpd-exp1>* does match T.

g) Condition c matches T iff c is true in the initial state of T.

Finally, we complete the definition of the meaning of a function, as follows:

Definition 8: A trajectory achieving a function

A trajectory T achieves a function F when the condition specified in C_F holds throughout T and G_F matches T.

Discussion and Summary

In this paper, we have presented CFRL, a language for specifying an expected function of a device and defined its semantics in terms of the type of behavior representation widely used in model-based qualitative simulation. The language allows one to explicitly state the physical context in which the function is to be achieved and to describe the function as an expected causal sequence of events. Since the concept of causal interactions among components is essential to the understanding of a function, the language allows explicit representation of causal interactions and constraints on such interactions.

CFRL is based on the work on Functional Representation (Sembugamoorthy & Chandrasekaran 1986), and it is a further extension of the work presented in (Iwasaki & Chandrasekaran 1992). We have extended the expressive power of the function specification languages

described in those papers and have provided a formal foundation for the semantics of the resulting language.

Franke (Franke 1991) also proposed matching design intent with simulated behavior. Unlike other work on functional representation, he focuses on representing the purpose of a design *modification* and not that of a device itself. He developed a representation scheme, called TED, in which he expresses the purpose for making a modification in a structure. TED's representation of a function can be a sequence (not necessarily a linear) of partial descriptions, which is matched against states in a sequence of qualitative states generated by QSIM. To prove that a function is achieved by a modification, he compares the behavior of the original structure and that of the modified structure.

Bradshaw and Young (Bradshaw & Young 1991) also represent the intended function in a manner similar to Functional Representation. They built a system called DORIS, which uses the knowledge generated by qualitative simulation for evaluating device behavior as well as for diagnosis and explanation.

The most important characteristic that distinguishes our work from those by Franke and by Bradshaw and Young is the central role causal knowledge plays in CFRL. We conjecture that causal relations are an essential part of functional knowledge, and that representation of functional knowledge must allow explicit description of the causal processes involved. Furthermore, verification of a function must ascertain that the expected causal chain of events take place, since the satisfaction of the functional goal alone does not necessarily indicate that the device is functioning as intended.

Because the semantics of CFRL is defined in terms of matching between a behavior and a functional specification, the language is immediately useful for the purpose of behavior verification. We have designed and implemented an algorithm that verifies a behavior produced by the DME system with respect to a function specified in CFRL as defined in this paper. Initial testing of the algorithm has included verifying the functional specifications of the EPS as given above. Care must be taken in designing such an algorithm to assure that exponential search is not required to find a match between a trajectory and a CPD. We are currently in the process of analyzing the computational complexity of the problem and our algorithm.

We expect formal functional specifications to have many uses throughout the life cycle of a device (Iwasaki, et al 1993). For example, in the early stages of the design process, designers often do "top down" design by incrementally introducing assumptions about device structure and causality relationships. Such design evolution could be expressed as incremental refinements of a CFRL functional specification. DME could assist a designer in this functional refinement process by assuring that each successive specification is indeed a refinement of its predecessor so that any device that satisfies the refinement also satisfies the predecessor.

References

Bradshaw J.A.; and Young R.M. 1991. Evaluating Design Using Knowledge of Purpose and Knowledge of Structure. IEEE Expert April.

Crawford J.; Farquhar A.; and Kuipers B. 1990. QPC : A Compiler from Physical Models to Qualitative Differential Equations. In Proceedings of the Eight National Conference on Artificial Intelligence.

Forbus K.D. 1984. Qualitative Process Theory. Artificial Intelligence 24.

Forbus, K. D. 1989. The Qualitative Process Engine. In *Readings in Qualitative Reasoning about Physical Systems*. Weld, D. S., and de Kleer, J. Eds. Morgan Kaufmann.

Franke D.W. 1991. Deriving and Using Descriptions of Purpose. IEEE Expert April.

Iwasaki Y.; and Simon H.A. 1986. Causality in device behavior. Artificial Intelligence 29:3-32.

Iwasaki Y.; and Low C.M. 1991. Model Generation and Simulation of Device Behavior with Continuous and Discrete Change. Technical Report, KSL, Dept. of Computer Science, Stanford University.

Iwasaki Y.; and Chandrasekaran B. 1992. Design Verification through Function and Behavior-Oriented Representations : Bridging the gap between Function and Behavior. In Proceedings of the Second International Conference on Artificial Intelligence in Design, Pittsburgh.

Keuneke A. 1989. Machine Understanding of Devices; Causal Explanation of Diagnostic Conclusions. Ph.D. thesis, Laboratory for AI Research, Dept. of Computer & Information Science, The Ohio State University.

Lockheed Missiles and Space Company. 1984. doc #D889545A, SE-23, Vol. 5.

Sembugamoorthy V.; and Chandrasekaran B. 1986. Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems. In Kolodner J.L. and Riesbeck C.K. (editors), Experience, Memory and Reasoning, Lawrence Erlbaum Associates, Hillsdale, NJ.

IJCAI-93

How Things Are *Intended* to Work: Capturing Functional Knowledge in Device Design

Yumi Iwasaki Richard Fikes Marcos Vescovi

Knowledge Systems Laboratory
Stanford University
701 Welch Road, Bldg C
Palo Alto, CA 94304

B. Chandrasekaran

Laboratory for AI Research
The Ohio State University
217 B, Bolz Hall, 2036 Neil Avenue
Columbus, OH 43210-1277

Abstract[1]

When designing a device, the final product of the design process is usually considered to be a physical specification of a device. However, the design of the causal mechanism underlying the physical specification, i.e. how the device is intended to work to achieve its function, is a product just as important as the physical specification, if not more. Capturing this knowledge of causal mechanism is necessary in order to understand the physical specification of the device as well as to evaluate and refine the specifications during the design process. Despite the importance of such knowledge, existing CAD tools do not support its explicit representation or manipulation. We describe a design support system under development in which knowledge of both the causal mechanism and the physical structure of a device being designed is explicitly represented and manipulated. The system allows the designer to provide functional specifications at various levels of abstraction in a language called CFRL (Causal Functional Representation Language). The CFRL specification acquired from the user enables the system to evaluate the physical specification as it is being developed in order to provide useful feedback to the designer. Furthermore, functional specifications provide an important basis for recording the engineer's design rationale.

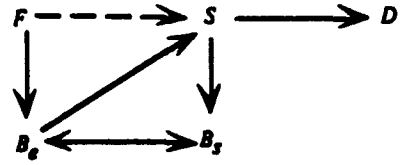
1 Introduction

Understanding *how* a device works requires understanding the causal sequences of events that achieve the function of the device. Books on how man-made or natural "devices" work are filled with drawings of structures of devices accompanied by explanations such as the following example of an aneroid barometer [Macaulay, 1988]:

As the air pressure falls, the spring pulls the side of the capsule outward. The arm rises, causing the rocking bar to slacken the chain. The hairspring unwinds, moving the pointer counter-clockwise until the chain is pulled taut.

Such explanations refer to parts of the structure and describe how it works in terms of the sequence of causal interactions among the parts that lead to the desired result. In other words, such explanations and the drawings they accompany describe the device structure along with the *conceptual causal mechanism* underlying the structure.

When designing a device, the final product of the design process is usually considered to be a physical specification of a device. However, the design of the causal mechanism underlying the physical specification is a product just as important as the physical specification, if not more. Gero makes this point clear in his model of the design process shown in Figure 1 [Gero, 1990]. Gero argues that, except in a trivial design problem, a structure is not generated directly from the requirements. In his model of design synthesis, a specification of the expected behavior is generated from the requirements[2], and the physical structure is generated from the expected behavior. In that model, the expected behavior corresponds to the expected causal sequence of interactions, i.e. how the device is to work.



B_e: Expected behaviors F: Functions (i.e. requirements)

B_s: Actual behaviors D: Design description

S: Structure

→ Transformation

⇢ Occasional transformation

↔ Comparison

Figure 1: Model of the design process [Gero, 1990]

[1]The research by the first three authors is supported in part by the Advanced Research Projects Agency, ARPA Order 8607, monitored by NASA Ames Research Center under grant NAG 2-581, and by NASA Ames Research Center under grant NCC 2-537. Chandrasekaran's research is supported by the Advanced Research Projects Agency by means of AFOSR contract F-49620-89-C-0110 and AFOSR grant 89-0250.

[2] Gero calls the requirements "functions" and denotes them with F in Figure 1. However, in this paper, we will use the term "requirements" to avoid confusion since we consider the term "function" to also include what he calls "expected behavior".

Capturing this knowledge of how the device is to work to achieve its function is important in order to understand the physical specification of the device as well as to evaluate and refine the specifications during the design process. One can even view the design of a conceptual causal mechanism as driving the generation of a physical specification. Despite the importance of such knowledge, existing CAD tools do not facilitate its explicit representation or manipulation.

According to Gero, the design process involves the following types of activities to manipulate the four types of information shown in Figure 1.

Formulation: Transforming requirements to expected behavior. $F \rightarrow B_e$

Synthesis: Transforming expected behavior to a structure. $B_e \rightarrow S$.

Analysis: Deriving behavior from a structure. $S \rightarrow B_s$

Evaluation: Comparing the predicted and expected behaviors. $B_s \leftrightarrow B_e$

Reformulation: Reformulating the expected behavior based on evaluation results. $(S, B_s) \rightarrow B_e$

Description: Producing a description of the designed structure. $S \rightarrow D$

A comprehensive design support system needs to support multiple iterations through all those steps, since any non-trivial design problem will require many such iterations. We are extending our Device Modeling Environment (DME) system [Iwasaki and Low, 1991] to provide such support and to enable explicit representation and manipulation of knowledge of both the causal mechanism and the physical structure of a device being designed. The system allows the designer to provide functional specifications at various levels of abstraction in a language called CFRL (Causal Functional Representation Language). The CFRL specification acquired from the user enables the system to evaluate the physical specification as it is being developed in order to provide useful feedback to the designer. Furthermore, functional specifications provide an important basis for recording the engineer's design rationale.

This paper describes the use of CFRL in the extended version of DME. We first describe CFRL and then illustrate the capabilities of the new system by presenting a hypothetical design scenario. We conclude with a discussion of our current status and related work.

2 CFRL

CFRL is a formalism we have developed for representing the functions and expected behavior of a device. It allows one to represent knowledge of the functions that a device is intended to achieve and also of the sequence of causal interactions among its components that lead to achievement of the functions. In CFRL, the function of the overall device is described first and the behavior of each component is described in terms of how it contributes to the function.

We have provided the language with a well-defined semantics in terms of a behavior representation widely used in model-based qualitative simulation [Vescovi et al, 1993], and we have used CFRL as the basis for a functional verification program which determines whether a behavior achieves an intended function.

In this section, we give a brief overview of CFRL. CFRL is fully described in [Vescovi et al, 1993]. Before describing CFRL, we present the device that we will use throughout the paper as an example. The device is the electrical power system (EPS) aboard an Earth-orbiting satellite [Lockheed, 1984]. A simplified schematic diagram of the EPS is shown in Figure 2. The main purpose of the EPS is to supply a constant source of electricity to the satellite's other subsystems. The solar array generates electricity when the satellite is in the sun, supplying power to the load and recharging the battery. When the satellite is in a shadow, the battery supplies power to the load. Since the battery can be damaged when it is charged beyond its capacity, the charge current controller opens the relay when the voltage exceeds a threshold to prevent the battery from being over-charged.

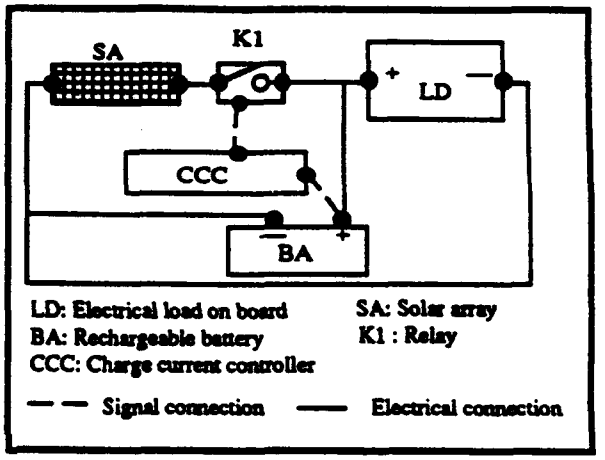


Figure 2: Electrical Power System

CFRL allows one to explicitly state the physical context in which a function is to be achieved and to describe a function as an expected causal sequence of events. Since the concept of causal interactions among components is essential to the understanding of a function, the language allows explicit representation of causal interactions and constraints on such interactions. Figure 3 shows an example of the representation of an EPS function.

Formally, a function F is defined as a triplet (D_F, C_F, G_F), where:

D_F denotes the device of which F is a function.

C_F denotes the context in which the device is to function.

G_F denotes a description of the functional goal to be achieved.

The notion of a device function assumes some physical context in which the device is placed, and C_F is a specification of such a context. C_F consists of two parts, a set of objects and a set of conditions on those objects. The conditions must hold throughout a behavior in order for the function to be satisfied by the behavior.

G_F, the goal to be achieved by the function, is represented as a Boolean combination of *Causal Process Descriptions* (CPDs). Each CPD is an abstract description of expected behavior in terms of a causal sequence of events. The abstracted behavior is represented as a directed graph in which each node describes a state and each arc describes a temporal and (optionally) a causal relation between states.

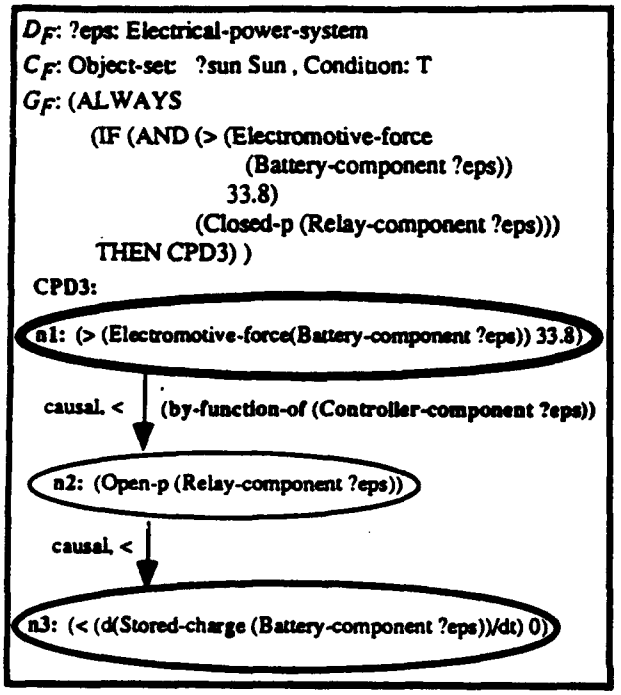


Figure 3 : EPS Function in CFRL.

A node specifies a condition on a state. The condition is a logical sentence that must hold in a state of the world at some time using the variables defined in D_F and C_F. The arcs in a CPD are directed and specify temporal and causal relations among nodes. An arc has the following attributes:

causal-flag: An indicator of whether the relationship between the states described by the source and destination nodes is causal.

temporal-relation: =, <, or ≤, indicating the temporal relation between the states described by the source and destination nodes. = means that the states described by the two nodes are to be the same state, < means the state described by the source node must strictly precede the state described by the destination node, and ≤ means the state described by the source node must either be the same as or precede the state described by the destination state.

causal-justification: If an arc is "causal", one can attach a justification for the causal relation. A justification takes the form of a Boolean combination of the predicates **by-function-of** (<component>) and **with-participation-of** (<component>). Those predicates are used to specify the participation of a device component in a particular portion of a causal process.

Because the semantics of CFRL is defined in terms of matching a functional specification to a device behavior, the language is immediately useful for design verification. In Figure 4, we show an example of matching a CPD to an EPS behavior generated by a qualitative simulator [Iwasaki and Low, 1991]. The left portion of Figure 4 shows part of an EPS function. (The notation has been changed slightly for legibility.) The right portion shows part of a predicted behavior for EPS. Since the IF part becomes true at state *s4*, *s4* is matched against node *n1*. Nodes *n2* and *n3* match with *s5* and *s6*. The temporal constraints on the arcs are clearly satisfied since $s4 < s5 < s6$. The way this portion of the behavior was simulated shows the existence of the following causal path involving the controller and the relay:

Voltage-battery > 33.8 *[s4]* ⇒ Turn-signal-on *[s4]*
⇒ Signal-control = on *[s5]* ⇒ Relay-open *[s6]*.

Thus, as required by the arcs in the CPD, the functioning of the controller plays a role in causing the condition specified in node n2 to become true and the functioning of the relay plays a role in causing the condition specified in node n3 to become true.

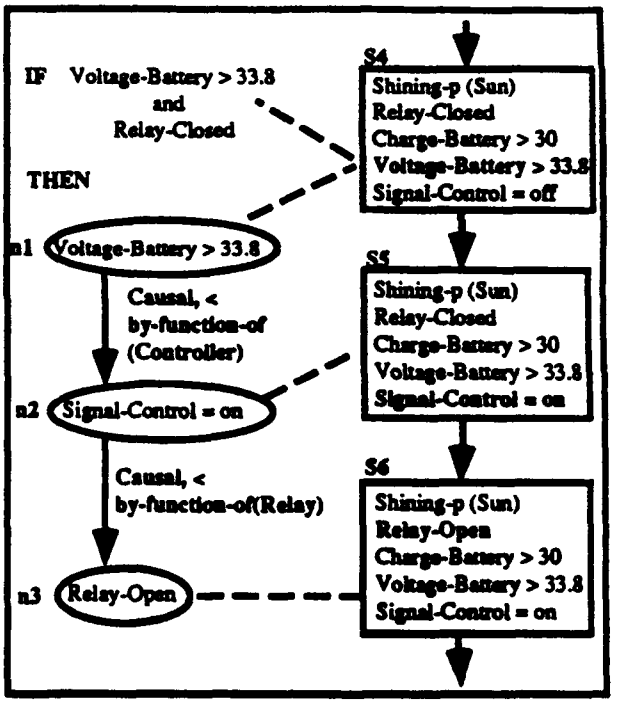


Figure 4: Example of matching a CPD to a behavior.

3 A Design Support System

A comprehensive design support system needs to support multiple iterations through all the steps of design, including formulation, synthesis, analysis, evaluation, reformulation, and description, since any non-trivial design problem will require many such iterations. Furthermore, even though the requirements are the input in Gero's model, requirements are rarely completely known a priori. Even known requirements often change during the design process, a provision not allowed for in most formal models of the

design process. During the process of generating and evaluating alternative designs, the designer may discover unforeseen consequences of given requirements or features of the environment that have effects on the performance. As a consequence, the requirements are likely to change many times. A practical design environment must support such incremental acquisition and evolution of the requirements. Our goal is to develop a design support system that can facilitate all these activities by:

- Providing languages for describing both device structure and functional specifications at various levels of abstraction.
- Supporting incremental acquisition of both constraints that must be satisfied by the device and properties of the environment in which the device is expected to operate.
- Providing analysis tools for evaluating the design through simulation and for verifying that the simulated behavior satisfies the functional specification.

We view the design process as one of parallel refinement of two types of specifications: physical and functional. The two specifications are closely linked in many ways. The functional specification drives development of the physical specification by providing the goals to be achieved. The functional specification refers to parts of the physical specification in order to explain how different components are to interact with each other to achieve the overall function. The physical specification must include at least all the components that are referenced in the functional specification and all connections that enable them to interact. Analysis of the behavior of the physical design in turn may suggest additional functions that are needed. Finally, the functional specification provides an important part of the rationale for the physical specification as well as the evaluation criteria for design.

The following section describes the design environment through a hypothetical design scenario.

3.1 Hypothetical Design Scenario

The given task is to design an electrical power supply (EPS) for an Earth orbiting satellite. The top-level goal is expressed in CFRL as shown in Figure 5.

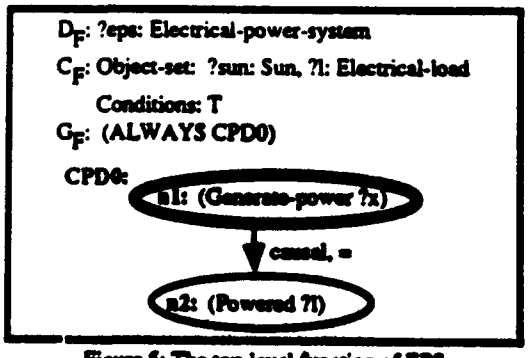


Figure 5: The top-level function of EPS

It says that the power supply generates power, supplying electricity to the load. The physical specification at this point, shown in Figure 6, is very simple. It shows that there is an electrical load connected to a power-supply.

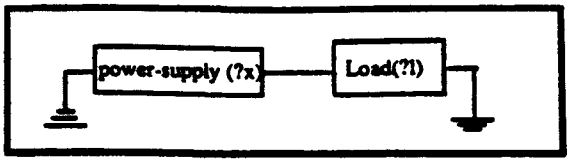


Figure 6: Top-level physical specification of EPS

The designer first decides to refine the power-supply component in Figure 6 by selecting a specific class of devices as a power supply. The system, using its component library, displays the following options:

Power-supply options: battery, solar array, thermal generator

The user selects "battery". The physical specification is refined to include a battery as the power supply. The user wants to see the behavioral implications of this selection. The systems performs qualitative simulation of this abstract physical specification using its generic knowledge of batteries. The simulation results, as shown in Figure 7, indicate that the battery will run out of power at some unspecified time *t*, after which the load will not be supplied power.

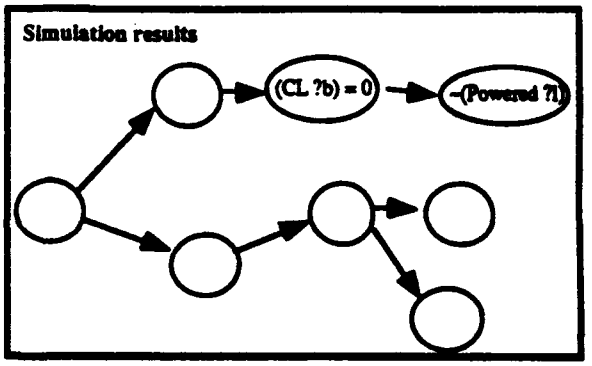


Figure 7: Qualitative simulation result

The system's verification module compares this predicted behavior with the top-level functional specification and informs the user that the function is not satisfied since the battery fails to supply power after some time. Furthermore, causal analysis by the verification module reveals that the failure to provide power causally depends on the capacity of the battery as well as the power demand by the load. These analysis results prompt the user to refine the functional specification by changing *"ALWAYS"* to *"for $t < 20$ years"* and by adding the constraint that the electrical load is *100 watts* and constant.

The user now asks the system to describe types of batteries that can satisfy these constraints. The system, using its battery knowledge base, describes possible types of batteries that can meet these requirements and lists their attributes, such as the capacities they must have. One of the listed attributes is weight, which the system computes will be between *500 kg* and *30,000 kg* depending on the type of battery chosen. The user decides that this is unacceptable for use in a satellite. She adds a new constraint that the

weight of the EPS must be less than *50 kg* and rules out using a battery as the power supply.

The user now returns to the list of alternatives supplied by the system and selects a generator as the power supply. The component knowledge base contains knowledge about the a priori requirements of each type of component for its operation. In the case of generators, the requirements include the availability of some type of fuel such as oil, coal, gas, or uranium. Since the system does not know enough about the environment in which the EPS will operate to determine if any such fuel is available, the system displays this requirement, which leads the user to specify the environmental constraint that there is no fuel and to rule out use of a generator.

Similarly, the user considers solar arrays and is presented with the requirement that sun light be available. The user adds the environmental constraint that sun light is available only for 70 minutes out of every 100 minutes and rules out solar arrays.

Since none of the options for a power supply presented by the system fulfills the functional goal, the designer decides to combine batteries and solar arrays, so that the solar array can recharge the battery as well as provide power while the sun light is available, and the battery can provide power when sun light is not available. She modifies the functional and physical specifications accordingly. She further refines the physical specification by choosing a nickel-cadmium battery. These steps produce the specifications and constraints shown in Figure 8a and 8b.

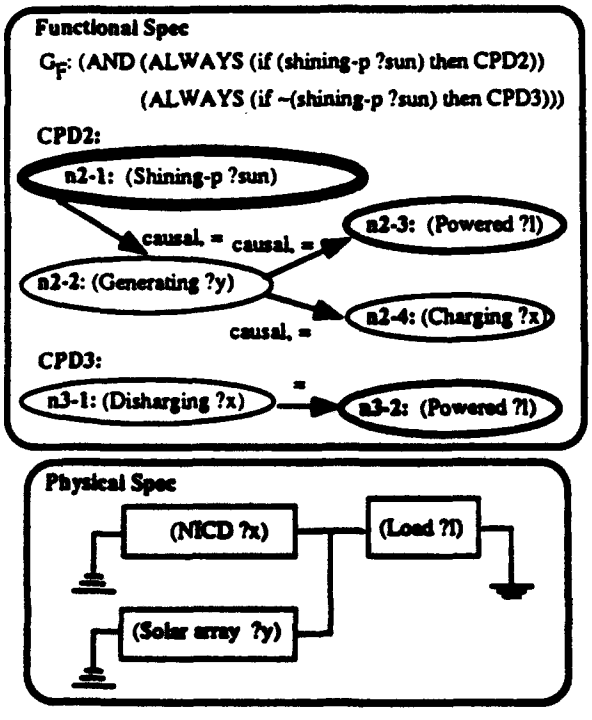


Figure 8a: Refined functional and physical specifications

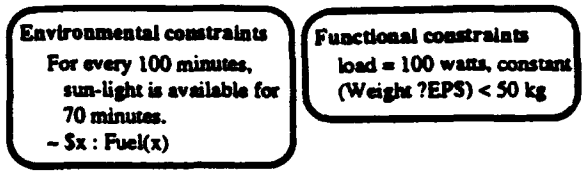


Figure 8b: Refined constraints

At this point, the designer decides to check the design by simulating its behavior. Predicted behavior indicates the possibility of the battery heating up and eventually becoming damaged. Noticing the battery heating up reminds the user that the scientific equipment on board the satellite should not be exposed to high temperature. She adds the functional constraint that the temperature of the EPS should not exceed *40° C*. Causal analysis of the states in which the battery heats up and becomes damaged reveals that overcharging of the battery is the cause.

The designer modifies the design again to include a controller, whose behavior she specifies to be opening the connection between the solar array and the rest of the circuit when the battery becomes fully charged and closing it otherwise. This new design is shown in Figures 9a and 9b. It also shows the functional specification is refined to include the function of the controller. The lower part of the functional specification shows the behavior of the controller as specified by the designer.

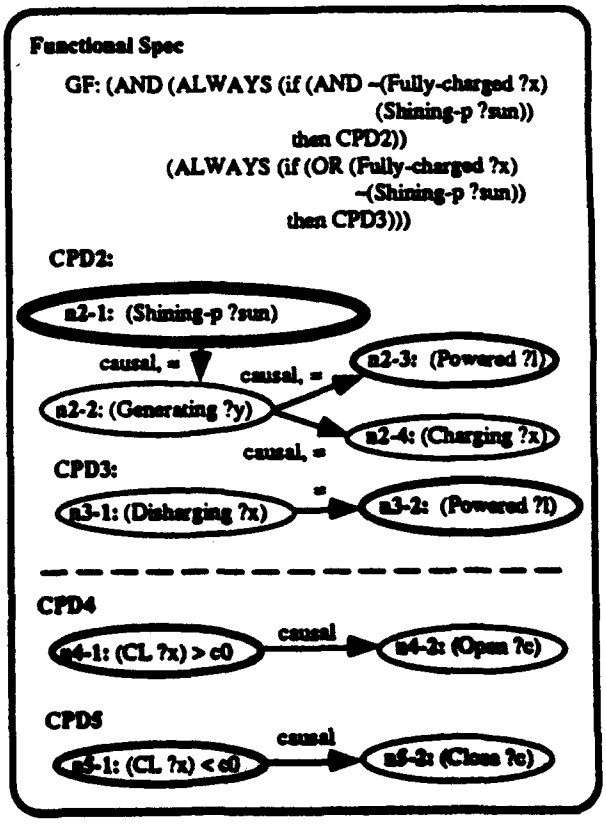


Figure 9a: Functional specification containing a controller

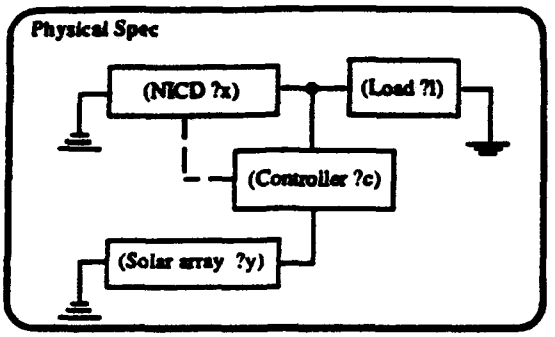


Figure 9b: Physical specification containing a controller

Simulating this new design reveals chattering behavior of the controller. The designer modifies the threshold for closing the controller in CPD5 to *c1* such that *c1* < *c0* to eliminate chattering.

3.2 Summary

To summarize, the design support system described in this section has the following features:

- Allows specification at various levels of abstraction of the physical structure and the intended functionality of the device being designed. The functionality is represented using CFRL and describes what the device is to achieve and how.
- Aids analysis of the design by providing simulation facilities which can automatically formulate a simulation model of a given design and predict its behavior. At an early design stage, simulation can be performed qualitatively to uncover possible undesirable behaviors. The system also provides causal analysis of the undesirable behaviors to help refine the design.
- Aids evaluation of the design by a verification facility that compares a predicted behavior against the functional specification in CFRL to determine whether the predicted behavior achieves the desired function.
- Aids in selection of components by presenting alternatives and their characteristics, using its component library indexed by function.
- Allows incremental acquisition and refinement of various design requirements, including functional and environmental constraints.

The final products of a design process using this design environment include physical and functional specifications as well as a complete list of requirements and a description of the operating environment. The functional specification should include a detailed description of what the device is expected to do, how it is expected to do it, and under what circumstances. Furthermore, the history of the design process will be maintained in order to enable post-facto reconstruction of the design rationale.

4 Discussion and Conclusion

In this paper, we have described a design support environment which allows explicit representation and refinement of functional specifications along with phys specifications. CFRL is used to express the intended function of the device to be designed and how the function is to be achieved through causal interactions of its components. Knowledge of the intended function of the device allows the system to evaluate the design of the physical structure during the design process to provide useful feedback to the designer even at an early stage of the design.

The described design support system is being implemented as an extension of the existing DME system. DME already contains model formulation and simulation facilities, a device structure editor, and an explanation facility [Gruber and Gautier, 1993]. An efficient algorithm for automatically formulating an appropriate simulation model for a given query has been developed and implemented [Iwasaki and Levy, 1993]. We have also implemented a behavior verification program based on CFRL [Vescovi et al, 1993].

As Gero points out, designing the expected behavior, i.e., what the device is to achieve and how, is an important part of the entire design process. Furthermore, in practical design problems, the requirements are not fully known initially but they are modified, augmented, and refined during the design process partly in response to analysis results of the design being developed. Therefore, a design environment must support such incremental acquisition of requirements as well as functional specifications.

A design support system needs to maintain a history of the design process, including development of the functional and physical specification, acquisition of requirements, and analysis performed of intermediate designs, including the desirable or undesirable behaviors discovered which led to further refinement of the design. Such a record can be used later to reconstruct not only rationale for the design of the physical structure, but also rationale for requirements and functional specifications.

There has been significant previous work on both representation of function and its use in reasoning about physical devices. CFRL is based on the work on Functional Representation [Sembugamoorthy and Chandrasekaran, 1986], and it is a further extension of the work presented in [Iwasaki and Chandrasekaran, 1992]. We have extended the expressive power of the language described in those papers, and have provided a formal foundation of the semantics of the language to make possible its use for verification of design. Franke [1991], and Bradshaw and Young [1991] also represent the intended function in a manner similar to Functional Representation. An important characteristic that distinguishes CFRL from the work of Franke, Bradshaw, and Young is the role causal knowledge plays in CFRL and its use in verification. Our work is based on the conjecture that causal relations are an essential part of functional knowledge, and that explicit representation and verification of such relations is important in ascertaining that the design achieves the intended goal.

Lee and Lai [1991] make a useful distinction between two types of design rationale: (1) a record of the exploratory activity of the design team, and (2) an account of how the designed artifact satisfies expected functionalities. The design support system described in this paper captures the latter type of rationale. It is a step towards realizing the view of functional representation as design rationale

discussed by Chandrasekaran et al [1993]. As such, the system we have described concentrates mostly on acquiring information about the structural and behavioral aspects of the device to be designed, but does not handle the full range of possible types of design rationale. There has been much work on development of systems for design rationale capture [Klein, 1993; Fischer et al, 1991]. In contrast to our work, most of them concentrate on capturing the second type of rationale in the form of argumentation structure (pros and cons) for individual features of the design. Few offer a representation formalism expressive enough to capture detailed functional knowledge of how the design is intended to achieve its goal, nor do they treat the development of the functional specification as a distinct activity in the design process. We believe that a comprehensive design support system must be able to represent and reason about both types of information, and we intend to extend our system to handle the full range of information involved in device design.

Acknowledgements

We would like to thank Steven Cousins, William Fung, and Hiang-Kwee Ho for their contributions to the discussions during which the extensions to DME described in this paper were refined.

References

[Bradshaw and Young, 1991] J.A. Bradshaw and R.M. Young. Evaluating Design Using Knowledge of Purpose and Knowledge of Structure. IEEE Expert, April 1991.

[Chandrasekaran et al, 1993] B. Chandrasekaran, Ashok Goel and Yumi Iwasaki. Functional Representation as Design Rationale. IEEE Computer, 26(1): 48-56, January 1993.

[Fischer et al, 1991] G. Fischer et al. Making Argumentation Serve Design. Journal of Human Computer Interaction, 6(3-4):393-419, 1991.

[Franke, 1991] David W. Franke. Deriving and Using Descriptions of Purpose. IEEE Expert, April 1991.

[Gero, 1990] John Gero. Design Prototypes: A Knowledge Representation Schema for Design. AI Magazine, 11(4):26-36, winter 1990.

[Gautier and Gruber, 1993] Patrice Gautier and Thomas Gruber. Generating Explanations of Devices Behavior Using Compositional Modeling and Causal Ordering. In Proceedings of the Eleventh National Conference on Artificial Intelligence, Washington D.C., July 1993. American Association for Artificial Intelligence.

[Iwasaki and Chandrasekaran, 1992] Yumi Iwasaki and B. Chandrasekaran. Design Verification through Function and Behavior-Oriented Representations: Bridging the gap between Function and Behavior. In Proceedings of the Second International Conference on Artificial Intelligence in Design, Pittsburgh, 1992.

[Iwasaki and Levy, 1993] Yumi Iwasaki and Alon Levy. Automated Model Selection for Simulation. In the Procedings of the Seventh International Workshop on Qualitative Reasoning about Physical Systems, Orcas Island, WA, 1993.

[Iwasaki and Low, 1991] Yumi Iwasaki and Che Meng Low. Model Generation and Simulation of Device Behavior with Continuous and Discrete Change. Technical Report KSL-91-69, Dept. of Computer Science, Stanford University, 1991.

[Klein, 1993] Mark Klein. Capturing Design Rationale in Concurrent Engineering Teams. IEEE Computer, 26(1):39-47, 1993.

[Lee and Lai, 1991] J. Lee and K-Y. Lai. What's in a Design Rationale? Journal of Human Computer Interaction, 6(3-4):250-280, 1993.

[Lockheed, 1984] Lockheed Missiles and Space Company. SMM Systems Procedure for Electrical Power Subsystem. doc #D889545A, SE-23, Vol. 3, 1984.

[Macaulay, 1988] David Macaulay. The Way Things Work. Houghton Mifflin Company, Boston, 1988.

[Sembugamoorthy and Chandrasekaran, 1986] V. Sembugamoorthy and B. Chandrasekaran. Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems. In Kolodner J.L. and Riesbeck C.K. (editors), Experience, Memory and Reasoning, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[Vescovi et al, 1993] Marcos Vescovi, Yumi Iwasaki, Richard Fikes and B. Chandrasekaran. CFRL: A Language for Specifying the Causal Functionality of Engineered Devices. In Proceedings of the Eleventh National Conference on Artificial Intelligence, Washington D.C., July 1993. American Association for Artificial Intelligence.

Appears in Computational Intelligence, 8(2), 1992, pp. 216-222

QP is More Than SPQR and Dynamical Systems Theory: Response to Sacks and Doyle

B. Chandrasekaran
Dept of Computer & Information Science
The Ohio State University
217B, Bolz Hall
2036 Neil Avenue
Columbus, OH 43210-1277

Ph 614-292-0923
Fax 614-292-9021
Email: Chandra@cis.ohio-state.edu
August 7, 1991

1 Where I Agree

I have followed, from a distance born of partly convergent and partly divergent goals, the research that has gone on in the name of "Qualitative Physics"(QP). The term QP is normally taken to mean reasoning about the physical world. A good deal of this work, however, has concentrated on prediction of behavior of physical configurations for which equational models from Physics can be written down in a tractable way (in contrast to, say, complex biological systems). Another part of this work has gone on in the domain of modeling functions and malfunctions of *devices*, and causal processes that participate in them. There has also been a body of work that has been called "naive physics," an attempt at modeling the commonsense knowledge of the physical world.

Sacks and Doyle's paper (hereinafter S&D) is very useful in bringing to light the difficulties with in one type of QP research, namely, prediction of behavior of certain types of physical systems. While I find the technical points made by S&D seem to be instructive overall, I think that the vision that they outline for a "new" QP needs to be significantly broadened.

Let me first review what I take to be the main technical point of the paper. For the behavior prediction task, the physical system is modeled as a vector of state variables of interest. Modeling includes specification, for each variable, of how a change in that variable affects other variables, specifically what changes in other variables follow. (For the kinds of systems considered, these relations are acausal, but if one part of the relation is taken to be the cause, the other is the effect. For example, in Newton's Law, $F = m\ a$, if the force is taken as the cause, the acceleration can be viewed as the effect, and vice versa.) The physical system is characterized by this (causal) model, which I shall call M. The behavior prediction task is to calculate or infer the values of all the dependent variables as changes in some of the independent variables are initiated.

When the variables are real numbers and the relationship between the changes in values of the variables is known completely and is given as a differential, the situation is fully char-

acterized by a set of simultaneous differential equations. Classical Physics teaches us how to set up such differentials for many physical systems. The theory of differential equations provides the calculus by which behavior can be predicted. This is all well-established as part of applied physics and mathematics.

However, the version of the prediction problem that QP theory attempts to solve is one where the input is known only qualitatively, or when the physical system itself is only qualitatively characterized. Under these conditions, we need new techniques by which behavior can be predicted, and several investigators have proposed such techniques (which S&D label SPQR techniques). As S&D show, while there are differences between the proposed techniques, all of them represent the relations between variables in some qualitative form (signs of changes around a "normal" value or monotonic relations). (That is, M — we can now call it M_{qual} — contains only qualitative relations between state variables.) There are corresponding proposals for calculi for prediction of behavior. These calculi share a style of inference in which the changes in the values of independent variables are propagated using the relations in M_{qual} until, to the extent the underlying ambiguities allow, the effects on all the variables are obtained.

S&D show that the calculi that have been offered in QP research do not perform particularly well. They claim that if one has a qualitative equation, then qualitative analysis using knowledge of advanced dynamical analysis can produce better results.

2 Why Do the QP Calculi Have Trouble?

What characteristic of the QP methods give them this disadvantage? S&D blame the qualitative language itself, that it misses relevant distinctions. Of course missing some distinctions comes with the territory for qualitative languages. But there is another reason for the difficulties faced by the QP calculi: the sequence of inferences mirrors the sequence of local causal interactions of the variables as described by the relations in the qualitative model, M_{qual}. That is, prediction is performed by literally "simulating" the system using the relations in M_{qual}. Sack's dynamical system analysis is not restricted by this property, it uses "global" techniques which directly generate qualitative analyses of the solution space. These techniques in some sense use much more knowledge of the mathematics of dynamical systems than is available in M_{qual} or the QP calculi. (A similar situation pertains to the causal ordering analysis of Iwasaki and Simon [Iwasaki and Simon, 1986] versus that by deKleer and Brown. The latter arrives at the causal ordering by following a series of local causal transitions, while the former is a global analysis of dependencies.) Others have recognized the importance of this property of the QP calculi: in [Forbus, 1988, Kuipers, 1986] similar observations about the distinction between information flow in the analysis and the flow of causality in the system are made.

There is also an additional problem in the QP calculi: M_{qual} only contains information about the specific physical system, the relations between variables are at one level of abstraction, and the calculi are relatively impoverished in inferential power. All human reasoning takes place with the benefit of substantial background knowledge about other abstractions and generalization rules. For example, a human might be able to use the knowledge that

reaching the same state again and again means that the system is in a cyclic state, and may make the inferential jump, even without advanced mathematical knowledge about dynamical systems.

The QP calculi seem to have been designed under some implicit constraints, namely, that they display some of the perceived properties of human reasoning about the physical world: that humans often appear to combine causal relations recursively, and in cases where they have the structure of the physical system available, trace the topology of the physical system to follow the "flow of causality." I will argue that QP techniques should aim to use the heuristic power of human reasoning even more, while employing the power of formal analysis to clearly defined subproblems where such techniques are needed. Thus the issue is broadened to include: What should the connection of QP research be to human commonsense knowledge and reasoning about the physical world? Is Newtonian physical modeling sufficient for QP, or necessary for all the goals of QP? If one were only interested in producing a technology that assists in reasoning about the physical world, can one develop this technology without to some degree being concerned with human commonsense knowledge and reasoning methods? My concern is to ensure that qualitative physics research has a significant place not only for mathematically sophisticated analysis techniques as S&D propose, but also for a whole spectrum of issues concerning the sources of the power in human reasoning about the physical world.

3 Human qualitative reasoning about the physical world

A trained physicist and an unschooled man-on-the-street start with a common ontology and a shared cognitive architecture. The physicist learns, and may add to, a specialized ontology as well, and acquires a number of modeling and analytical techniques. We need to sort out these distinct types of knowledge about the physical world that come into play in human reasoning.

1. A *commonsense ontology* which predates and is in fact used by modern science: space, time, flow, physical objects, cause, state, perceptual primitives such as shapes, and so on. The commonsense ontology also comes with some terms that are given specific technical meanings by science, but in general the terms in this ontology are experientially and logically so fundamental that scientific theories are built on the infrastructure of this ontology. Early work in QP had as a main goal elaboration of such an ontology ([Hayes, 1979, Forbus, 1984] are examples). Even today, a good deal of QP research grapples with the development of ontologies for different parts of commonsense physical knowledge.

2. The *scientific ontology* is built on the commonsense ontology (and often gives specific technical meanings to some of the terms in it, such as "force"). Additional concepts and terms are constructed. Some of these are quite outside commonsense experience (examples are "voltage," "current," and "charm of quarks").

3. *Compiled causal knowledge.* People compile causal expectations partly from direct experience and partly by caching some results from earlier problem solving. Which causal expectations get stored and used is largely determined by the relevance of the causes and effects to the goals of the problem solver. There is a more organized form of causal knowledge

that we build up as well: models of *causal processes*. By process model I mean a description in terms of temporally evolving state transitions, where the state descriptions are couched using the commonsense and scientific ontologies. For example, we have commonsense causal processes such as "boiling," or specialized ones such as "voltage amplification," "the business cycle," and so on. These are not neutral, agent-independent, process descriptions, but ones in which the qualitative states that participate in the description have been chosen based on abstractions of interest to the agent. In particular, such descriptions are couched in terms of possible intervention options on the world to affect the causal process, or observations to detect the process. Forbus' processes [Forbus, 1984] and my and my colleagues' work on functional representations [Sembugamoorthy and Chandrasekaran, 1988, Keuneke, 1991, Goel, 1989, Sticklen and Tufankji, 1991] are examples concerned with the development of representations for causal processes.

When the process model is based on pre-scientific or unscientific views, we have naive process models (such as models of sun rotating around the earth, or of exorcism of evil spirits). Many pre-scientific process models are not only quite adequate, but are actually simpler and more computationally efficient than the the scientific ones, for everyday purposes.

These process descriptions are great organizing aids: they focus the direction of prediction, help in the identification of structures to realize desired functions in design [Goel, 1989], and suggest actions to enable or abort the process.

4. *Mathematical equations embodying scientific laws and expressing relations between state variables.* These equations themselves are acausal, and any causal direction is given by additional knowledge about which variables are exogeneous.

4 Some of the Things That A New QP Should Include

It is generally agreed, including by S&D, that a QP theory or framework should provide support for three components of reasoning about the physical world: modeling, prediction and control. In fact, a weakness of their paper is that they pay only lip service to the problem of modeling and fail to show why or how dynamic analysis will help solve that and the control problems. With the recent exception of SPQR calculi, quite a bit of the work in QP research is concerned with the development of ontologies, which are directly relevant to the modeling problem. Since I expect other respondents to outline precisely how the QP field is paying attention to these problems, I will concentrate on those aspects of the problem unlikely to be emphasized by them.

4.1 Modeling

All modeling is done in the context of goals to be accomplished, i.e., states to be achieved or avoided in the world. The heart of the modeling problem is to map from goals to tractable representations. Compiled causal knowledge (see discussion in Section 3) plays an essential role in identifying aspects of the physical situation and perspectives that need to be represented. The causal process models can be used to identify states that should be represented and reasoned about. Given a physical situation and goals, causal processes

whose results are relevant to the achievement of the goals are retrieved and used as a guide in modeling the situation. The aggregation levels (when dealing with populations) [Weld, 1986], the abstractions, the approximations and the concepts in the representation are all jointly determined by the physical situation, the goals, and the rich storehouse of causal process knowledge that expert reasoners possess. Progress in modeling requires progress in ontology development and causal process descriptions.

4.2 Prediction

The power of experts in prediction comes, not from wholesale formalization of the problem in terms of Physics and subsequent qualitative or other simulation (as much of current QP work tends to present the problem), but by the use of a substantial body of compiled causal knowledge in the form of causal process descriptions to hypothesize states of potential interest. Further, the state variables participating in causal relations may not all be continuous, and hence, even in principle, not all problems of prediction can be formulated as analysis of dynamical systems. For example, a substantial part of our causal knowledge is about nominal variables ("vacations relax people," "lack of support causes objects to fall"). Simon [Simon, 1991] describes a causal ordering scheme which works with such variables, but, as a rule, the SPQR models and the dynamic system analysis techniques work only with state variables which are continuous.

Humans, in their everyday life, rarely predict behavior in the physical world by generating a long series of causal chains, certainly not a series of inferences that can be called "sound." The reasons for this are brought out clearly by QP work: ambiguities proliferate rapidly. If you ask someone what will happen if a ball is thrown at a wall, very little of the sequence of predictions is the result of application of scientific laws of motion. Rather, a short series of causal sequences are constructed from compiled causal knowledge, instantiated to the specific physical situation. Two important sources of power that are available for human experts in generating successor states and handling ambiguities are discussed next.

Compilation of consequences

If we ask someone, "what will happen if I throw a rock at the glass window?" that person is likely to say, "the window might break." A number of such causal fragments, compiled from experience or from earlier problem solving episodes, are stored as part of our causal knowledge about domains of interest. The ambiguity ("might") is OK, since the goal of qualitative prediction is typically not accuracy or certainty, but identification of an interesting possibility that may be investigated more thoroughly if needed.

Handling ambiguity

Ambiguities in causal simulation are often handled not on the basis of what effect *will* happen, but on the basis of what *might* happen that may help or hurt the explicit or background goals of the problem solver. Thus, when there is more than one successor state in simulation, the state that is related to goals of interest is chosen. In the example involving

the glass window, suppose a person was standing on the other side of the glass window, and you saw some one about to throw a rock at the window. You would most likely attempt either to stop the rock throwing or alert the person standing at the window. You would not be paralyzed with the ambiguities in prediction: the rock may not really hit the window, the window may not shatter, the rock may miss the person, the rock or glass fragments may not draw blood, and so on. Not only the commonsense world, but engineering reasoning is also full of such goal-driven ambiguity handling. For example, in design analysis, one might use this form of ambiguity handling to identify the possibility that a component will make its way into a dangerous state. Of course, once this possibility is identified, quantitative or other normative methods can be used in a selective way to verify the hypothesis. This heuristic role of qualitative behavior prediction is often ignored in QP research in favor of concerns about completeness and soundness of predictions.

There is often no reason to make a complete and ambiguity-laden prediction of a physical situation, if small and possibly retractable changes can be made to the physical world, and changes directly noted. In fact, this is just another instance of using the real world as a computational aid [Chapman, 1990], and avoiding long chains of reasoning based on complex symbolic reasoning models.

In engineering and scientific reasoning, whenever reasoning about consequences reaches a point where relatively precise answers are needed for choices to be made — and only then — the situation can be selectively modeled and analytical methods of varying degrees of complexity and precision can be employed. The models that are formed reflect the problem solving goal that is current, and typically represent only a small slice of the physical system.

4.3 Control

For control of reasoning (or the selection and deployment of reasoning methods, the so-called problem of "intelligent control of methods"), we will need a theory of task analysis for various tasks and domains. Such a task analysis will delineate alternative methods for a problem, and the subtasks that each method generate. The properties and knowledge requirements for each method could also be identified as part of such a task analysis, so that choices of methods are done with due respect to the needs of goals and knowledge availability. For example, a decision might be made about whether the current goal can be met by using stored causal process descriptions, by obtaining more accurate information from the real world, or by performing calculations. An engineer might be able to use order-of-magnitude reasoning in some cases to decide that safe levels of current will obtain in a circuit. In other cases, predictive reasoning (with some type of ambiguity resolution) might simply identify unsafe current level as a possibility. In the latter case, a detailed formulation of Kirchoff's laws for the relevant subsystem can be made and the equations solved. Thus, an item in the research agenda for any QP of the future will need to be the development of task structures for various tasks involved in QP. An example of such a task analysis is the task structure for design is given in [Chandrasekaran, 1990].

5 Summary

The QP community represents many diverse goals. Even if one were not especially interested in cognitive modeling and only cared about producing powerful technologies to help in reasoning about physical devices, QP can exploit representations and techniques that form part of human expertise.

The thrust of my discussion has been that if we are going to be doing a new QP, we might as well expand its scope to include these different sources of power in expert reasoning. While I agree with the points raised by S&D regarding the deficiencies of the current QP calculi and also with their argument for the importance and power of qualitative analysis of dynamical systems using advanced mathematical knowledge, the general problem of QP is larger that that of reasoning about systems for which Physics models can be readily made and which can be cast as dynamical systems of certain types. Many QP researchers have recognized this and have been hard at work on developing the ontologies and process descriptions needed for the modeling task. However, much QP research in modeling and reasoning is too concerned with step-by-step soundness at the expense of the heuristic power of much human reasoning. It misses the role played by the "situatedness" of human reasoning about the physical world: the background goals and the rich store of causal knowledge already possessed. It is not that one wants the results to be incorrect, but soundness of human causal reasoning is the result of interesting and effective hypotheses made in the first place, followed, if necessary, by a focused application of verification or refinement techniques.

Finally, we need to expand our consideration of media for representation and manipulation of concepts; for example, reasoning with pictorial representations for problems involving shapes seems to be a natural direction of research.

Acknowledgements: I thank R. Bhaskar, Tom Bylander, John Josephson, Leo Joskowitz, Dale Moberg, Hari Narayanan, Sunil Thadani, and Michael Weintraub for comments on my first draft. I think they sort of agree, but I'm sure, not completely. Daniel Weld, as editor of the responses, made very helpful comments as well. I also acknowledge the support of DARPA contract F-49620-89-C-0110 in the preparation of this article.

References

[Chandrasekaran, 1990] Chandrasekaran, B. (1990). Design problem solving: A task analysis. *AI Magazine*, 11(4):59–71.

[Chapman, 1990] Chapman, D. (1990). Vision, instruction and action. Technical report, MIT AI Lab, Cambridge, MA.

[Forbus, 1988] Forbus, K. (1988). Qualitative physics: Past, present and future. In Shrobe, H., editor, *Exploring Artificial Intelligence*, pages 239–296. Morgan Kaufmann, San Mateo, CA.

[Forbus, 1984] Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, 24:85–168.

[Goel, 1989] Goel, A. K. (1989). *Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving*. PhD thesis, The Ohio State University.

[Hayes, 1979] Hayes, P. (1979). The naive physics manisfesto. In Mitchie, D., editor, *Expert Systems in the Micro-Electronic Age*. Edinburgh University Press, Edinburgh.

[Iwasaki and Simon, 1986] Iwasaki, Y. and Simon, H. (1986). Causality in device behavior. *Artificial Intelligence*, 29:3–32.

[Keuneke, 1991] Keuneke, A. (1991). Device representation: The significance of functional knowledge. *IEEE Expert*, 6(2):22–25.

[Kuipers, 1986] Kuipers, B. J. (1986). Qualitative simulation. *Artificial Intelligence*, 29(3):289–338.

[Sembugamoorthy and Chandrasekaran, 1988] Sembugamoorthy, V. and Chandrasekaran, B. (1988). Functional representation of devices and compilation of diagnostic problem-solving systems. In Kolodner, J. and Reisbeck, C., editors, *Experience, Memory, and Reasoning*, pages 47–73. Lawrence Erlbaum Associates.

[Simon, 1991] Simon, H. A. (1991). Nonmonotonic reasoning and causation: Comment. *Cognitive Science*, 15(2):293–300.

[Sticklen and Tufankji, 1991] Sticklen, J. and Tufankji, R. (1991). Utilizing a functional approach for modeling biological systems. *Advances in Mathematics and Computers in Medicine (to appear)*.

[Weld, 1986] Weld, D. (1986). The Use of Aggregation in Causal Simulation. *Artificial Intelligence*, 30(1).

Architecture of intelligence: The problems and current approaches to solutions

B. Chandrasekaran and Susan G. Josephson
Laboratory for AI Research, The Ohio State University, Columbus, OH 43210, USA

We propose as a working hypothesis a Separability Hypothesis which posits that one can factor off an architecture for cognition from a more general architecture for mind, thus avoiding a number of philosophical objections that have been raised about the 'strong AI' hypothesis. Using a coin-sorting machine as an example, we discuss a range of positions on representations and argue that, for many purposes, the same body of matter can be interpreted as bearing different representational formalisms. We then propose that one way to understand the diversity of architectural theories is to make a distinction between deliberative and subdeliberative architectures. The search for *one* architectural level which will explain all the interesting phenomena of cognition is likely to be futile. There are a number of levels that interact, and this interaction makes explanation in terms of one level quite incomplete.

Dimensions for thinking about thinking

A major problem in the study of intelligence and cognition is the range of—often implicit—assumptions about what phenomena these terms are meant to cover. Are we just talking about cognition as having and using knowledge, or are we also talking about other mental states such as emotions and subjective awareness? Are we talking about intelligence as an abstract set of capacities, or as a set of biological phenomena? These two questions set up two dimensions of discussion about intelligence. After we discuss these dimensions we will discuss information processing, representation, and cognitive architectures.

Dimension 1. Is intelligence separable from other mental phenomena?

When people think of intelligence and cognition, they often think of an agent being in some knowledge state, that is, having thoughts, beliefs. They also think of the underlying process of cognition as something that changes knowledge states. Since knowledge states are particular types of information states the underlying process is thought of as information processing. (We will discuss this in more detail later in the paper.) However, besides the knowledge states, mental phenomena also include such things as emotional states and subjective consciousness. Under what conditions can these other mental properties also be attributed to artifacts to which we attribute knowledge states? Is intelligence separable from these other mental phenomena?

It is possible that intelligence can be explained or simulated without necessarily explaining or simulating other aspects of mind. A somewhat formal way of putting this *Separability Hypothesis* is that the knowledge state transformation account can be factored off as a homomorphism of the mental process account. That is: If the mental process can be seen as a sequence of transformations: $M_1 \rightarrow M_2 \rightarrow \ldots$, where M_i is the complete mental state, and the transformation function (the function that is responsible for state changes) is F, then a subprocess $K_1 \rightarrow K_2 \rightarrow \cdots$ can be identified such that each K_i is a knowledge state and a component of the corresponding M_i, the transformation function is f, and f is some kind of homomorphism of F. A study of intelligence alone can restrict itself to a characterization of K's and f, without producing accounts of M's and F. If cognition is in fact separable in this sense, we can in principle design machines that implement f and whose states are interpretable as K's. We can call such machines *cognitive agents*, and attribute intelligence to them if they achieve goals. However, the states of such machines are not necessarily interpretable as complete M's, and thus they may be denied other attributes of mental states.

For example, Searle[1] holds that a computer program that successfully translates from Chinese to English cannot be said to 'understand Chinese', even though it is behaviorally intelligent in this task. In our terminology, we would attribute to the program various appropriate knowledge states. Searle's objection can be formulated as the claim that 'understanding' is a subjective property that goes beyond merely being in the corresponding knowledge state, and thus the

program can be denied that attribute.

However, other researchers claim that intelligence cannot be separated from other mental phenomena. Such a claim is often made from two opposite perspectives. Most people in artificial intelligence (AI) and cognitive science say that intelligence and other aspects of mind are inseparable because the other mental aspects (subjectivity, emotional states, etc.) are simply 'emergent' properties of certain kinds of complex agents with knowledge states. If this is the case, the knowledge state account, and with it an account in terms of information processing, will be a sufficient basis for explaining and building minds. From this perspective, explanation of the phenomena of intelligence and cognition will also turn out to be explanation of the full range of mental phenomena. By the same token, it is assumed that artificial agents that can be plausibly interpreted as solving problems, achieving goals, and performing reasoning will also have emotional states and subjective consciousness attributable to them.

The second perspective from which intelligence is taken to be inseparable from other mental phenomena holds that there is no coherent way to factor off a knowledge state process account from a mental state process account. There is only one mental process. That is, from this point of view, the categorical difference between different attributes of mental states is affirmed, but the Separability Hypothesis is denied. We can talk about knowledge components of mental states, but mental processes have no 'subprocesses' which only have to do with information processing. In this view, the only way to explain or build an intelligence is to solve the problem of explaining or building a mind. Thus only agents which have the totality of what we call 'mind' will be able to perform as truly successful problem solvers across the whole range of situations deemed to require intelligence.

Edelman[2,3] has argued that information processing is not the appropriate way to talk about cognition. Instead he proposes that the basic mechanisms of the brain are the selection of successful neural pathways in response to interactions with the world. The processes that underlie this neuronal path selection resemble Darwinian evolutionary processes. Cognitive phenomena, in his view, cannot be separated and understood in information processing terms, since cognitive states are simply aspects of more general brain states, and the basic brain mechanisms are not information processes.

Dimension 2: Functional versus biological

The second dimension in discussions about intelligence involves the extent to which we need to be tied to biology for understanding intelligence. Can intelligence be characterized abstractly as a functional capability which just happens to be realized more or less well by some biological organisms? If it can, then study of biological brains or of human psychology is not logically necessary for a theory of cognition and intelligence, just as enquiries into the relevant capabilities of biological organisms are not needed for the abstract study of logic and arithmetic or for the theory of flight. Of course, we may learn something from biology about how to practically implement intelligent systems, but we may feel quite free to substitute non-biological (both in the sense of architectures which are not brain-like and in the sense of not being constrained by considerations of human psychology) approaches for all or part of our implementation. Whether intelligence can be characterized abstractly as a functional capability surely depends upon what phenomena we want to include in defining the functional capability, as we discussed. We might have different constraints on a definition that needed to include emotion and subjective states from one that only included knowledge states. Clearly, the enterprise of AI deeply depends upon this functional view being true at some level, but whether that level is abstract logical representations as in some branches of AI, Darwinian neural pathway selections as proposed by Edelman, something intermediate, or something physicalist is still an open question.

Newell holds a functional view of intelligence. According to Newell[4], intelligent agents can be abstractly characterized by their goals, their knowledge and the Principle of Rationality. That is, when we attribute intelligence to an agent in some behavior, we are attributing to that agent a goal, a body of knowledge, and a capability, at least in that instance of behavior, of applying knowledge relevant to the goal to decide what to do. It is important to note that all of this is *attribution*. Newell calls a description of an agent in these terms a *Knowledge Level* description. Knowledge Level descriptions view the agent as being in a knowledge state, and the Principle of Rationality as the abstract characterization of how the agent changes knowledge states. (Attributing knowledge and goals to an agent is similar to taking an *intentional stance* towards agents that Dennett[5] has proposed.)

There is no claim that knowledge is internally represented explicitly, and in just the same propositional units as in the attribution, or that explicitly inferential processes are operating. Newell defines the functionality of intelligence as the ability of an agent to realize the knowledge potential inherent in its Knowledge Level description. For Newell the important character of intelligence is the agent's ability to make full use of the knowledge attributed to it, not the amount or the specifics of the agent's knowledge. Even humans are only an approximation to the ideal intelligence so

defined. In this perspective, biological evolution will be seen as operating in the direction of better and better approximation to this sort of intelligence through the evolution of more complex knowledge state representations (of the sort that finds its culmination in human language) which are capable of supporting open-ended deliberation and the application of knowledge to new goals.

So, with Newell, we have a functional characterization of intelligence which is independent of biology. But Newell goes on to propose an architecture which is inspired by one biological instantiation, the human cognitive apparatus. This architecture is similar to the human one in that it has a long-term memory and a deliberative architecture similar to the one that in his view characterizes human cognition. But, because it is a functional architecture, it goes beyond the biological in many ways. For example, the ideal architecture always retrieves the relevant knowledge, unlike the human version which often fails to remember. Further, the functional architecture is based on digital computer-like symbol structures. For Newell, it does not matter if the human brain is literally such a computer. All that matters is that the kind of computer-like symbol structures can support the functionality needed. Further, the architecture that is proposed by Newell as a possible one for AI is just one among many possible realizations of the abstract functional capability specified in his definition of intelligence.

In general, functional characterizations end up using aspects from very different levels of descriptions of biological mind. For example, the connectionists want to be biological enough to include some of the smooth concept learning done by humans, and an architecture based on some abstract properties of what they take to be the information processing of brains, but their orientation is not so biological as to demand wet neurons and neuronal chemistry. Searle wants to be biological enough to demand the inclusion of the subjective awareness of being in a knowledge state (which is how we interpret his claim that a translator who follows the algorithm does not really 'understand Chinese') that humans have, but he thinks that it is most likely the chemistry of the brain that is responsible for it, and thus a pure information processing account will not succeed. Edelman wants to be biological enough to include the way in which organisms' brains, in his view, do not use pre-made internal labels (which he takes to be the characteristic property of information processing). Since his theory of pathway selection itself is stated as an abstract mechanism, presumably artifacts could be constructed which implement that abstract architecture without any further reference to biology. Connectionists (Rumelhart *et al.*[6]) and Edelman want to be biological enough to understand the common heritage between animals and humans, while traditional AI researchers stop their biological commitment to characterizing intelligence as using knowledge to reason and achieve goals (since they take humans to be doing that). Thus, all such proposals pick out some interesting aspect from biological phenomena. They then proceed to formulate a functional model that includes the selected aspect. After this, real biology is no longer logically necessary. Whether any of these proposals would lead to the production (or explanation) of mentality in total, or almost circularly, produce only those aspects of mentality that are included in the functional definition, is obviously an open question.

Coin-sorters and knowledge states

In this article we will take the Separability Hypothesis as a working hypothesis. At this point, *for all practical purposes AI (and cognitive science) can be considered the study of those regularities of mind that have information-processing explanations.* We will assume that it is a worthwhile enterprise to concentrate on phenomena in which knowledge states of the agent seem to play the central role. Further we will focus on processes that account only for generation and transformation of such knowledge states. Now this might appear to be a commitment to information processing so strong that many interesting theories will be ruled out. However, we will argue that the knowledge state account is very flexible, and can even be applied to situations where there is no explicit information processing in the conventional sense. To illustrate this we will use the example of a coin-sorter for coins of USA.

Analysis of a coin-sorter

Let us suppose that we have a black box coin sorter in front of us, and we want to describe its behavior computationally. All we see is that the coins are put into the top of the coin sorter, and then they come out through one of four slots at the bottom, with all the dimes coming out of the slot designated the dime slot, and the quarters coming out of the slot designated the quarter slot, and so on. Let us assume we have four types of AI theorists: a logician, someone who is committed to algorithms alone as the language in which to formulate AI theories, a connectionist and a physicalist, i.e. one who claims that the appropriate explanation of the coin-sorter should be in terms of its physics, not representations.

Logic system coin sorter. The logician proposes that the machine's behavior can be understood in terms of

four logical axioms, one for each coin. A set of measurements is made on each of the coins. Perhaps diameter, weight and thickness are the coin's important features for this purpose. Each coin type is characterized by a logical formula of predicates involving the measurements. For example, the axioms for each of the four types of coins will indicate what combination of weight, thickness and diameter characterize that coin type. The logician claims that the behavior of the machine can then be characterized by a theorem-proving decision procedure that attempts to prove each of the theorems for each coin that is inserted, followed by a mechanism that places the coin into that slot corresponding to the theorem that was proved.

Note that this language enables us to argue about different theories about what is being measured by the sorter. Someone could watch the behavior of the coin sorter and assert that the machine is not using information about the weight and diameter of the coins, but rather about, say, its color and metallic content. They could propose an alternative axiom system in terms of color and metallic content. Each such axiom system is a different content theory expressed in the logic formalism.

Further, the formalism can be used to evaluate these alternate theories and test them experimentally. We can use logical inference to draw out the consequences of each proposal. One hypothesized content theory might predict that a given foreign coin, say an Indian rupee, will come out of the quarter slot, while another might predict that the rupee will come out of the penny slot. We can then test to see which hypothesized content theory most accurately describes the decision-making process within the black box by putting the rupee in and seeing which slot it is placed at.

Notice that the usefulness of the logic formalism has two levels. On one level, we can use the formalism to describe different content theories, e.g. the theory that the coins are being sorted by color versus one that they are being sorted by weight. We can use the inference machinery that comes with logic to derive consequences of different axioms and test one theory of representational content against another. For this purpose, there is no need to commit oneself to how the insides of the sorter work in any detail, except that information of certain types is being used to make decisions of certain types. We are simply using logic to reason about the agent, much as it is used in computer science to reason about the correctness of a computer program written in some other language than logic. We are using logic to give a *Knowledge Level* description of the system.

The second use of logic may be to model, or carry out, internal processing. For example, the coin-sorter might actually have dedicated Prolog chips inside actually implementing the theorem provers. The coin-sorter might literally work by actuating an arm that places the coins in the slots as soon as the results of the theorem provers are available.

Decision tree coin-sorter. The second theorist observes the coin-sorter and announces that its behavior can be described by a decision tree. In a decision tree machine, there is an initial decision made between two groups, e.g. between the group consisting of the nickel and the quarter, and the group consisting of the dime and the penny. For each of the groups, at the next point in the tree, an additional decision is made to make a choice among subgroups, and this is repeated until each leaf node corresponds to one of the elements of the original group. We now have a decision tree. In the coin-sorter example, we would only need two levels in the tree. The criteria for the decisions at each node are given in the form of rules involving values of measurements made on the coin.

Again, we can use the formalism as a descriptive device, or as a commitment to a certain internal processing. For example, as a descriptive device, the decision tree still enables us to propose different content theories, not only about what aspects of the coins are measured as in the logic case, but also about what sets of decisions are made before what decisions. In this sense, what was left as a feature of internal processing in the use of logic for external description, namely some aspect of control strategy, is actually now made part of the external description of the device. The axiom system made no commitment to control. This expresses the difference between a Knowledge Level account and a program level account.

On the other hand, similar to the logic case, one can imagine microprocessors actually implementing the decision tree algorithm, using the measurements to make the choices in the tree, and activating the coin-placing mechanism appropriately when a leaf node is reached.

Connectionist network coin-sorter. The connectionist claims that what is really going on in the coin sorter involves the same features, diameter, color, or whatever, as the other theories assumed, but these evidences are 'weighted' and combined as in a connectionist network. Different theories of representational content could still be represented by identifying the nodes with different types of measurement. How the information is used can be described by means of different weights and thresholds in the network. Intermediate abstractions may be captured by hidden units. The intermediate abstractions are combined with other intermediate abstractions and again weighted and higher level decision units are constructed. A specific output node is

identified for each coin. The 'energy' at the output nodes will be a function of how much evidence is coming through for the coin for which it stands. The output node corresponding to the largest activation will be chosen as the decision node.

Pretty much all the points we made about logic and decision trees can be repeated for this account as well. The connectionist framework can be used to describe content theories about what information is used, and to give an account of what evidence is combined in what proportion with what other evidence. Inferences about different content theories can be made and tested. At this level, no commitment needs to be made that the inside of the sorter is literally a connectionist machine. On the other hand, the connectionist network can be used as the internal information processor as well.

Voila: *Levers and holes!* Let us now open the coin-sorter and look at its inside. We see that as you put a coin in, it passes through levers and holes, all cleverly arranged such that the coin makes its way to the right slots. Clearly, the different weights and the sizes of the coin have different effects on the levers and the holes. There are no prologue chips or microprocessors or connectionist networks inside the black box, just mechanical parts. The physicalist, the one who does not believe in representations, smiles.

Does the sorter have a knowledge state interpretation?

In response to the question, 'How did the quarter end up in the slot named "quarter"?', two kinds of answers, both correct, can be given. In one, the answer would be physicalist: an account of the coin's movement through the inside of the sorter following the physical laws. In the other, the answer would be in terms of how the levers and holes 'use' information about the diameter and the weight and how the sorter 'decides' about the coin's direction of movement. Clearly, whoever designed it designed the sorter in such a way that there is a close mapping between the information story and the physical story. Because of this mapping, one can talk about the sorter being in various knowledge states. Of course, if the sorter that works by levers and holes has a consistent interpretation in terms of knowledge states, then certainly any sorter that actually had a chip proving theorems or implementing the decision tree algorithm or the connectionist network will also have a similar interpretation. That is, the knowledge state and information processing talk is applicable to all devices whose behavior has a decision-making interpretation, irrespective of how they actually work.

We can see that the logic account, the decision tree algorithm account and the connectionist account are all alternative languages in which to couch the information processing account. While all three frameworks can be used to describe information representation and processing, they are not all equivalent. Connectionism enables one to talk about 'softer' combination of information using real numbers, while logic enables us to talk about variables and quantification, and the language of algorithms enables us to talk about control strategies. However, our main point here is that they can all be used as frameworks for describing information representation and processing, and also for implementing information processing. In Newell's language, they can be used both as languages for the Knowledge Level and for the Symbol Level.

The coin-sorter is a simple device, but it illustrates the issues with respect to understanding biological brains. People take a whole range of stances on whether the brain is actually doing information processing on representations. Strong materialists argue that representationalist accounts of such systems are wrong, and the only scientifically acceptable causal story is at the level of the matter that composes the brain. Edelman is also against the information processing account, but his counter-proposal is in terms of an abstract pathway selection account, which is still an abstract functional architecture (i.e. no appeal to physical laws is made), though not an information processing one. Among those who agree that there is a causal story to be told at the level of representations, there are many divisions, but broadly, we can distinguish between connectionist style representations and discrete symbol structure representations. The moral of our analysis of the coin-sorter is that for explaining behavior which itself is couched in informational terms, the information processing account is useful as a stance to describe the biological brain.

Much of the argument in the field is a result of a confusion between two senses of being an information processor using representations. In one sense, when we ask whether the brain processes information we are really asking whether it is appropriate to ascribe informational activity to the brain and in the other sense we are literally describing what the brain or device actually does. Ascribing information processing is to take an information processing stance. For example we might ascribe information processing activity to the visual system on the grounds that it produces information about the world. This is the sense of information-processing we are using when we stand outside the brain and look at behavior and ascribe an information-processing structure to the behavior that we see. When we look at a black box coin sorter as a decision maker and work out a model of its input/output behavior, we are ascribing information processing to it.

However, taking an informational stance whereby we

ascribe information processing to a device (or brain) does not commit us to that device literally processing information, or using representations, in the specific medium in which the description is made. There *is* a fact of the matter about whether the information processing is being done in one medium or another. At some point the behavior of the sorter which employs a Prolog theorem prover will be different from that based on levers and holes. When the latter sorter malfunctions, the explanation may be given in terms of physical properties, such as a lever being jammed, while in the case of the former type of sorter, the explanation might be in terms of an error in the program in the chip or some hardware failure in the chip. (And in the case of the brain, in addition to the problem of failure modes, there are other issues where the medium becomes relevant: properties related to learning, are one example.) But for most purposes where people think that the issue is the medium of representation, the issue often turns out to be one that can be formulated at the Knowledge Level.

We can certainly ask similar questions about the brain. It is a matter of fact whether the brain is an information processor of the 'physicalist' type, one of the connectionist variety, or one that has Turing machine-like symbols. (Putnam[7] has argued that even whether a piece of matter is a Turing machine is just a stance, but we think that the consensus is that Putnam's argument does not really work, and that not all pieces of matter can be interpreted as a given Turing machine.) But as long as we are interested in aspects of the organism's behavior that have an informational flavor (such as decision-making), talk of information and its use is necessary in the analysis, just as it was in the case of the coin-sorter. Much of the criticism of the information processing view (from Edelman, e.g.) of information processing is based on a narrow view of what the information-processing talk commits one to. Conversely, many proponents of information processing explanations are also committed to such a narrow view, making far more commitments about internal processes than necessary.

In the rest of the article, we will adopt this broad sense of information processing or knowledge state account as a stance that is useful in describing agents to which we ascribe cognitive capacities.

Architectures for intelligence

We now move to a discussion of architectural proposals within the information processing perspective. Our goal is to try to place the multiplicity of proposals into perspective. As we review various proposals, we will present some judgements of our own about relevant issues. But first, we need to review the notion of an architecture and make some additional distinctions.

Form and content issues in architectures

In computer science, a programming language corresponds to a virtual architecture. A specific program in that language describes a particular (virtual) machine, which then responds to various inputs in ways defined by the program. The architecture is thus what Newell calls the fixed structure of the information processor that is being analysed, and the program specifies a variable structure within this architecture. We can regard the architecture as the *form* and the program as the *content*, which together fully instantiate a particular information-processing machine. We can extend these intuitions to types of machines which are different from computers. For example, the connectionist architecture can be abstractly specified as the set $\{\{N\}, \{n_1\}, \{n_0\}, \{\zeta_i\}, \{w_{ij}\}\}$, where $\{N\}$ is a set of nodes, $\{n_1\}$ and $\{n_0\}$ are subsets of $\{N\}$ called input and output nodes respectively, $\{\zeta_i\}$ are the functions computed by the nodes, and $\{w_{ij}\}$ is the set of weights between nodes. A particular connectionist machine is then instantiated by the 'program' that specifies values for all these variables.

We have made a distinction between an architecture, the form in which the architecture will accept content (the programming language form) and the content of the representation itself. When we explain specific types of cognitive phenomena, we will end by coming up with a complex budget of credit allocation: some aspects will be explained by the properties of the architecture (perhaps some timing phenomena, and also some aspects of learning), some will be explained by the sort of information that is involved in the content. Credit allocation in this manner is a tricky analytic task.

We also need to make an additional distinction between micro- and macro-architectures, a distinction that is especially useful for cognition. A bank of information processors of identical type connected in some way has a macro-architectural description in terms of the modules and their connections, while the entire system has a uniform micro-architectural description.

Many AI and cognitive science theories are really theories about the content of knowledge, or types of knowledge, needed for some task of interest, with minimal commitment to the architecture. Many debates in the field, which are ostensibly about the architecture, turn out to be about the types of knowledge. For example, Dreyfus[8] talks about 'What computers cannot do'. It turns out that he is opposed to the idea that intelligence can come out of a system that has a knowledge base which explicitly and exhaustively

represents world facts and relationships in some logical form. However, there are people within computational AI who have been making this point as well. For example, Schank[9] has argued that our knowledge is not in the above form of abstract facts at all, but rather in the form of experiences indexed and abstracted in various ways. Thus the issue, at least based on Dreyfus' arguments, is not what computers cannot do, but what certain kinds of knowledge representations cannot do. It may turn out that the kind of information that Dreyfus sees as necessary cannot be represented in computers either, but he does not make the arguments for this position.

We are now ready to give an overview of the issues in cognitive architectures. We will assume that the reader is already familiar in some general way with the proposals that we are discussing. Our goal is to place these ideas in perspective.

Intelligence as just computation

Until recently the dominant paradigm for thinking about information processing has been the Turing computer framework, or what has been called the discrete symbol system approach. Information processing theories are formulated as algorithms operating on data structures. In fact AI was launched as a field when Turing proposed in a famous paper that thinking was computation (the term 'artificial intelligence' itself was coined later). A natural question in this framework would be whether the set of computations that underlie thinking is a subset of Turing-computable functions, and if so, how the properties of this subset should be characterized.

Because of the technological nature of much of AI, only a small number of researchers have been concerned with intelligence in general. Most of the work consists of algorithms for specific problems that seem to require intelligence and that are practically important. Algorithms for diagnosis, design, planning, etc. are proposed, because these tasks are seen as important for an intelligent agent. But as a rule no effort is made to relate the algorithm for the specific task to a general architecture for intelligence. While such algorithms are useful as technologies and to make the point that several tasks that appear to require intelligence can be done by certain classes of machines, they do not give much insight into intelligence in general.

Architectures for deliberation

Historically most of the intuitions in AI about intelligence have come from introspections about human consciousness, specifically about what people perceived to be the relationships among conscious thoughts. We are aware of having thoughts which often follow one after another. These thoughts are mostly couched in the medium of natural language, but sometimes thoughts include mental images as well. When people are thinking for a purpose, say for problem solving, there is a sense of directing thoughts, choosing some, rejecting others, and focusing them towards the goal. Activity of this type has been called 'deliberation'. Deliberation, for humans, is a coherent goal-directed activity, lasting over several seconds or longer. For many people thinking is the act of deliberating in this sense. Activities in this time span should be contrasted with other cognitive phenomena, which, in humans, take under a few hundred milliseconds: real-time natural language understanding and generation, visual perception, being reminded of things, and so on.

Different kinds of theories about the architecture of the cognitive machine have been proposed depending upon what kinds of patterns among these thoughts the researchers have been struck by. Two groups of proposals about such patterns have been influential in AI theory-making: the *reasoning* view and the *goal–subgoal* view.

Deliberation as reasoning. People have for a long time been struck by logical relations between thoughts and have made the distinction between rational and irrational thoughts. Remember that Boole's book on logic was titled 'Laws of Thought'. Thoughts often have a logical relation between them: we think thoughts A and B, then thought C, where C follows from A and B. In AI, this view has given rise to an idealization of intelligence as rational thought, and consequently to the view that the appropriate architecture is one whose behavior is governed by rules of logic. In AI, McCarthy is most closely identified with the logic approach to AI, and ref. 10 is considered a clear early statement of some of the issues in the use of logic for building an intelligent machine. Researchers in AI disagree about how to make machines which display this kind of rationality. One group proposes that the ideal thought machine is a logic machine, one whose architecture has logical rules of inference as its primitive operators. These operators work on a storehouse of knowledge represented in a logical formalism and generate additional thoughts. For example, the Japanese Fifth generation project came up with computer architectures whose performance was measured in (millions of) *inferences* per second. The other group believes that the architecture itself (i.e. the mechanism that generates thoughts) is not a logic machine, but one which generates plausible, but not necessarily correct, thoughts,

and then knowledge of correct logical patterns is used to make sure that the conclusion is appropriate.

Historically rationality was characterized by the rules of deduction, but in AI, the notion is being broadened to include a host of non-deductive rules under the broad umbrella of 'non-monotonic logic'[11] or 'default reasoning', to capture various plausible reasoning rules. There is considerable difference of opinion about whether such rules exist in a domain-independent way as in the case of deduction, and how large a set of rules would be required to capture all plausible reasoning behaviors. If the number of rules is very large, or if they are context-dependent in complicated ways, then logic architectures would become less practical.

At any point in the operation of the architecture, many inference rules might be applied to a situation and many inferences drawn. This brings up the control issue in logic architectures, i.e. decision about which inference rule should be applied when. Logic itself provides no theory of control. The application of the rule is guaranteed, in the logic framework, to produce a correct thought, but whether it is relevant to the goal is decided by considerations external to logic. Control tends to be task-specific, i.e. different types of tasks call for different strategies. These strategies have to be explicitly programmed in the logic framework as additional knowledge.

Deliberation as goal–subgoaling. An alternate view of deliberation is inspired by another perceived relation between thoughts and provides a basic mechanism for control as part of the architecture. Thoughts are often linked by means of a *goal–subgoal* relation. For example, you may have a thought about wanting to go to New Delhi, then you find yourself having thoughts about taking trains and airplanes, and about which is better, then you might think of making reservations and so on. Newell and Simon[12] have argued that this relation between thoughts, the fact that goal thoughts spawn subgoal thoughts recursively until the subgoals are solved and eventually the goals are solved, is the essence of intelligence as a mechanism. More than one subgoal may be spawned, and so backtracking from subgoals that did not work out is generally necessary. Deliberation thus looks like search in a problem space. Setting up the alternatives and exploring them is made possible by the knowledge that the agent has. In the travel example above, the agent had to have knowledge about different possible ways to get to New Delhi, and knowledge about how to make a choice between alternatives. A long term memory is generally proposed which holds the knowledge and from which knowledge relevant to a goal is brought to play during deliberation. This analysis suggests an architecture for deliberation which retrieves relevant knowledge, sets up a set of alternatives to explore (the problem space), explores it, sets up subgoals, etc.

The most recent version of an architecture for deliberation in the goal–subgoal framework is Soar[4]. Soar has two important attributes. The first is that any difficulty it has in solving any subgoal simply results in the setting up of another subgoal, and knowledge from long term memory is brought to bear in its solution. It might be remembered that Newell's definition of intelligence is the ability to realize the knowledge level potential of an agent. Deliberation and goal subgoaling are intended to capture that capability: any piece of knowledge in long term memory is available, if it is relevant, for any goal. Repeated subgoaling will bring that knowledge to deliberation. The second attribute of Soar is that it 'caches' its successes in problem solving in its long term memory. The next time there is a similar goal, that cached knowledge can be directly used, instead of searching again in the corresponding problem space.

This kind of deliberative architecture confers on the agent the potential for rationality in two ways. With the right kind of knowledge, each goal results in plausible and relevant subgoals being setup. Second, 'logical rules' can be used to verify that the proposed solution to subgoals is indeed correct. But such rules of logic are used as pieces of knowledge rather than as operators of the architecture itself. Because of this, the verification rules can be context- and domain-dependent.

Another point to note is that one of the results of this form of deliberation is the construction of special purpose algorithms or methods for specific problems. These algorithms can be placed in an external computational medium and as soon as a subgoal arises that such a method or algorithm can solve, the external medium can solve it and return the results. For example, during design an engineer might set up the subgoal of computing the maximum stress in a truss, and invoke a finite element method running on a computer. The deliberative engine can thus create and invoke computational algorithms. The goal–subgoaling architecture provides a natural way to integrate external algorithms.

In the Soar view, long term memory is just an associative memory. It has the capability to 'recognize' a situation and retrieve the relevant pieces of knowledge. Because of the learning capability of the architecture, each episode of problem solving gives rise to continuous improvement. As a problem comes along, some subtasks are solved by external computational architectures which implement special purpose algorithms, while others are directly solved by compiled knowledge in memory, while yet others are solved by additional deliberation. This cycle makes the overall system increasingly more powerful. Eventually, most

routine problems, including real-time understanding and generation of natural language, are solved by recognition. (Recent work by Patten *et al.*[13] on the use of compiled knowledge in natural language understanding is compatible with this view.)

Deliberation seems to be a source of great power in humans. Why is not recognition enough? As Newell points out, the particular advantage of deliberation is distal access to and combination of knowledge at run-time in a goal-specific way. In the deliberative machine, temporary connections are created between pieces of knowledge that are not hard-coded, and that gives it the ability to realize the knowledge level potential more. A recognition architecture uses knowledge less effectively: if the connections are not there as part of the memory element that controls recognition, a piece of knowledge, though potentially relevant, will not be utilized in the satisfaction of a goal.

As an architecture for deliberation, the goal–subgoal view seems to us closer to the mark than the reasoning view. As we have argued elsewhere[14], logic seems more appropriate for justification of conclusions and as the framework for the semantics of representations than for the generative architecture.

AI theories of deliberation give central importance to human-level problem solving and reasoning. Any continuity with higher animal cognition or brain structure is at the level of the recognition architecture of memory, about which this view says little other than that it is a recognition memory. For supporting deliberation at the human level, long term memory should be capable of storing and generating knowledge with the full range of ontological distinctions that human language has.

Is the search view of deliberation too narrow?. A criticism of this picture of deliberation as a search architecture is that it is based on a somewhat narrow view of the function of cognition. It is worth reviewing this argument briefly.

Suppose a Martian watches a human in the act of multiplying numbers. The human, during this task, is emulating some multiplication algorithm, i.e. appears to be a multiplication machine. The Martian might well return to his superiors and report that the human cognitive architecture is a multiplication machine, but we know that the multiplication architecture is a fleeting, evanescent virtual architecture that emerged as an interaction between the goal (multiplication) and the procedural knowledge of the human. With a different goal, the human might behave like a different machine. It would be awkward to imagine cognition to be a collection of different architectures for each such task; in fact, cognition is very plastic and is able to simulate various virtual machines as needed.

Is the problem space search engine that has been proposed for the deliberative architecture such an evanescent machine? One argument against it is that it is not intended for a narrow goal like multiplication, but for all kinds of goals. Thus it is not fleeting, but always operational.

Or is it? If the sole purpose of the cognitive architecture is goal achievement (or 'problem solving'), then it is reasonable to assume that the architecture would be hard-wired for this purpose. What, however, if goal achievement is only one of the functions of the cognitive architecture, common though it might be? At least in humans, the same architecture is used to daydream, just take in the external world and enjoy it, and so on. The search behavior that we need for problem solving can come about simply by virtue of the knowledge that is made available to the agent's deliberation from long term memory. This knowledge is either a solution to the problem, or a set of alternatives to consider. The agent, faced with the goal and a set of alternatives, simply considers the alternatives in turn, and when additional subgoals are set, repeats the process of seeking more knowledge. In fact, this kind of search behavior happens not only with individuals, but with organizations. They explore alternatives, but we do not see a need for a fixed search engine for explaining organizational behavior. Deliberation of course has to have the right sort of properties to be able to support search. Certainly adequate working memory needs to be there, and probably there are other constraints on deliberation, but it does not have to be exclusively a search architecture. Just like the multiplication machine was an emergent architecture when the agent was faced with that task, the search engine is the corresponding emergent architecture for the agent faced with a goal and equipped with knowledge about what alternatives to consider. In fact, a number of other such emergent architectures built on top of the deliberative architecture have been studied earlier in our work on Generic Task architectures[15]. These architectures were intended to capture the needs for specific classes of goals (such as classification).

The above argument is not to deemphasize the importance of problem space search for goal achievement, but to resist the identification of the architecture of the conscious processor with one exclusively intended for search. The problem space architecture is still important as the virtual architecture for goal-achieving, since it is a common, though not the only, function of cognition.

Of course, that cognition goes beyond just goal achievement is a statement about human cognition. If we take a design perspective and seek to specify an architecture for a function called intelligence which itself is defined in terms of goal achievement, then

clearly we are free to design an architecture best suited for that purpose. A deliberative search architecture working with a long term memory of knowledge certainly has many attractive properties for this purpose as we have discussed in this section.

Architectures below deliberation

We made a distinction between cognitive phenomena that occur in under a few hundred milliseconds and those that evolve over longer time spans, and covered the latter under deliberation. We will call the architecture that handles the former phenomena *subdeliberative*. In deliberation, we have access to a number of intermediate states in problem solving. After you finished planning the New Delhi trip, I can ask you what alternatives you considered, why you rejected taking the train, and so on, and your answers to them will generally be reliable. You were probably aware of rejecting the train option because you calculated that it would take too long. On the other hand, we have generally no clue about how the subdeliberative architecture came to any conclusion. If you recognize someone after not having seen him for twenty years, and that person expresses surprise by asking, 'I have changed a lot in twenty years. How did you recognize me?', you may come up with something like, 'I bet it is your nose!', but you cannot be sure. You have no access to how your perception system actually recognized that person. Similarly, you may have your own theory of why you were reminded of something, but you have no special access to what went on during that reminding. Freud's model of mind had complicated unconscious processes working, and in fact, in this view, consciousness was often misled about the real content of these unconscious processes.

Many people in AI and cognitive science feel that the emphasis on complex problem solving as the door to understanding intelligence is misplaced, and that theories that emphasize rational problem solving only account for very special cases and do not account for the general cognitive skills that are present in ordinary people. This group of researchers focus almost completely on the nature of the subdeliberative architecture. There is also a belief that the subdeliberative architecture is directly reflected in the structure of the neural machinery in the brain. Thus, some of the proposals for the subdeliberative architecture claim to be inspired by the structure of the brain and claim a biological basis in that sense.

Alternative proposals. The various proposals differ along a number of dimensions: what kinds of tasks the architecture performs, degree of parallelism, whether it is an information processing architecture at all, and when it is taken to be an information processing architecture, whether it is a symbolic one or some other type.

With respect to the kind of tasks the architecture performs, we already mentioned Newell's view that it is just a recognition architecture. Any smartness it possesses is a result of good abstractions and good indexing, but architecturally, there is nothing particularly complicated. In fact, the good abstractions and indexing themselves were the result of the discoveries of deliberation during problem state search. Being smarter, from the Newell perspective, is done by converting more and more deliberative problems into stored recognition patterns through chunking. The real solution to the problem of memory, for Newell, is to get chunking done right: the proper level of abstraction, labeling and indexing is all done at the time of chunking. Theories of memory representation (such as Schank's) are in this sense content theories of indices and labels, not architectural theories. Such content theories of memory are not really in conflict with the Newell theory of deliberative architecture, since the latter merely gives a way for the content to come to be the way it is.

In contrast to the recognition view are proposals that see relatively complex problem solving activities going on in subdeliberative cognition. Minsky[16] originally proposed a specific architecture for memory based on frames, which are organized as a network of concepts, each of which contained prototypical information about the concept. Relatively complex procedures were embedded in these concepts. More recently, he has outlined a Society of Mind[17] architecture for cognition. Cognition in this picture is a communicating collection of modular agents, each of whom is simple, but capable of some degree of problem solving. For example, they can use the means-ends heuristic (the goal–subgoaling) feature of deliberation in the Soar architecture).

Deliberation has a serial character to it. Almost all proposals for the subdeliberative architecture, however, use parallelism in one way or another. Parallelism can bring a number of advantages. For problems involving similar kinds of information processing over somewhat distributed data (like perception), parallelism can speed up processing. Some problems that require explicit search if done serially can be done without search in a parallel architecture. For example, perception problems often involve evaluating a number of alternative interpretations and choosing the best. These alternatives can be simultaneously assessed in parallel and the best picked. Ultimately, however, additional problem solving in deliberation may be required for some tasks.

Within the school that views the subdeliberative architecture as representation-processing, there has been a debate about the medium in which information is represented. Turing computational architectures have

been the representational frameworks of choice for modeling deliberation. For subdeliberation, the same framework was used until connectionism came along. Connectionism replaced the explicit processing of symbolic tokens with a specific type of analog computation. The original connectionist proposal of the PDP type[6] were in some ways less powerful than Turing machines. For example, it had to face the criticism that that kind of computation cannot account for the systematicity and generativity of natural language which requires variable binding and symbols of some type[18], requirements which the Turing-computational framework can handle well. A number of ways of enlarging the connectionist frameworks to give them these capabilities have been proposed. Some involve using explicit symbols in connectionist representations (see for example, ref. 19), while others involve representations that have some of the properties of symbols without being symbols in the Turing-computational sense (see for example, ref. 20). In any case, most of these connectionist proposals are actually implemented and simulated in digital computers, and none of the functions that they compute are outside the Turing framework. The problem does not really seem to be with Turing computation *per se*, but rather the way in which Turing computation has been used in AI and cognitive science, namely as applications of inference on axiomatically represented world knowledge.

Connectionism has been evolving in a number of directions. A proposal that has been gaining currency is that the information processing of the brain is a dynamical system[21] defined by complex nonlinear differential equations. It has been claimed, for example, that chaos may be useful as a creative device for new states in a search[22], and that dynamic systems at criticality have the unbounded dependencies characteristic of context-sensitive grammars[23].

Edelman argues strongly against information processing theories of cognition on the ground that they require a prelabeled world of objects and relations, whereas biological organisms in his view discover patterns as regularities in their interactions with the world rather than start with prelabeled information. He also argues against connectionism since he thinks they require some form of prelabeled information as well. His architectural proposal is not couched as computation on representations, but as one in which successful neural pathways are selected in a process similar to Darwinian evolution. The selection is done in response to the physical interaction of the organism with the external world. This process results in neural structures which categorize the organism's interaction with the world, but these are not fixed logical categories, but flexible, constantly changing ones, to reflect the organism's continuing interaction. Edelman has proposed additional mechanisms by which these structures develop higher and higher order categorizations and coordinations.

The motivation behind connectionism and its offshoots is generally couched as opposition to symbolic computation, and Edelman argues against information processing, but, as we have argued earlier, the real opposition seems to be to the idea of a representational repertoire that corresponds to the theories of the external world of objects and relations that we conceptualize in our conscious models of the world. There is a widespread suspicion that AI and cognitive science have confused the externally visible constructions of mind (explicit knowledge of the world, grammars, etc.) as the raw material of mind. In this view, just because we seem to be using pieces of knowledge in our deliberation does not mean that this knowledge was represented in that form in memory. The phrase 'information processing' has been too closely associated with the view that what is inside the mind is much like what we seem to have in our consciousness. The opposing view is that whatever is inside us is not in the form of abstract statements of facts about the world, but rather is concretely tied to our interaction with the physical world, flexible, open-ended, and constantly changing with each interaction.

With this proviso accepted, we can take a representational stance towards connectionist networks as well as Edelman's selection machine. In that sense of attributed information or knowledge that we argued for in our discussion of the coin-sorter, Edelman's organism has knowledge and information. We can, from outside, watch an Edelmanian brain at some point in its evolution, and say things like, 'This organism knows about x, but not about y.' In the broad sense of information processing that we have been advocating, Edelman's organism is an information processing agent and its neural pathways represent knowledge. If knowledge of the world can be in the form of on-going abstractions of experience, which at the Knowledge Level, can be interpreted as partial, but increasingly more veridical, knowledge of the world, then all these approaches qualify as information processing theories.

Is there a 'right' architectural theory of subdeliberation? Later in the article we discuss how to place the various alternative proposals in useful relations to each other.

So far we have talked about the micro-architecture of the subdeliberative system. A few brief comments on macro-architecture are relevant. Fodor[24] has proposed the *Modularity Hypothesis* which asserts that there are separate modules for each of the perceptual modalities, the language modality and central cognition. That is, there is relatively little interaction between them until the perceptual and language modules have completed

their interpretation tasks. These interpretations are available in the working memory of deliberation. There is some debate about how much information flow is there from one modality to another during recognition, but there is general consensus that the degree of intermodality information flow is small in comparison with the information processing within each module.

Situated cognition. Real cognitive agents are in contact with the surrounding world containing physical objects and other agents. A new school has emerged calling itself the *situated cognition* movement which argues that traditional AI and cognitive science abstract the cognitive agent too much away from the environment, and place excessive emphasis on internal representations. The traditional internal representation view leads, according to the situated cognition perspective, to excessive amounts of internal representation and complex reasoning using these representations. Real agents simply use their sensory and motor systems to explore the world and pick out the information needed, and get by with much smaller amounts of internal representation processing. At the minimum, situated cognition is a proposal against excessive 'intellection'. In this sense, we can simply view this movement as making different proposals about what and how much needs to be represented internally. However, there are more radical versions of the movement in which any internal representation is denied. Specifically, the movement rejects the idea that knowledge is represented in the brain and retrieved as needed, but instead holds that knowledge is constructed by the agent in a complex interaction between neural processes and the external situation. '[Representations] are the *product* of interactions, not a fixed substrate from which behavior is generated'[25]. The reader will recognize that this view is close to that of Edelman. This constructivist view of knowledge is a major dividing line between traditional 'knowledge representation' view in AI and the situated cognition view. To take an example, schema theories in psychology and frame theories in AI have held that memory is organized in terms of schemas, stereotyped concepts or events. The newer view would hold that such schemas are actually constructed in response to the situation, not units of memory representation and organization[26].

In our discussions so far, we have presented two different views on internal representations. On the one hand, we have representations in the traditional AI sense of explicit encoding of facts and so on, and on the other hand, we also said that one can often take an external Knowledge Level stance towards the content of knowledge that is implied by an agent's behavior. The situated cognition perspective clearly rejects the former view with respect to internal (sub-deliberative) processes, but accepts the fact deliberation does contain and use knowledge. Thus the Knowledge Level description could be useful to describe the content of agent's deliberation. But the perspective emphasizes the issues relevant to the nature of the neural level descriptions and the processes which work with the external situation to construct the representations in deliberation.

The movement raises many important issues, but the solution to the problem of what sort of neural processes exist and how the interactive process constructs representation is still in the future.

Integrating the perspectives

An integrated view of problem solving. We briefly outline how the major components of the cognitive architecture work together in the solution of complex problems. The agent is embedded in the physical world, receives sensory information, and acts on the world. Deliberation is the central co-ordinating architecture, and its working memory can contain both symbolic and imagistic data, constructed out of long term representations in response to the goal at hand, as the situated cognition movement proposes. Memory can be viewed at the Knowledge Level as containing this information, but this talk should not mislead one into thinking that the information that is in working memory was in that form in long term memory (see our discussion on situated cognition). The agent also has action repertoires which can be thought of as a form of memory, but information representational talk is much less appropriate for describing them.

The degree of abstract problem solving required depends on the kind of goal. Many goals can be simply solved by means of one or more of the action repertoires, with little mediation from anything that one might call problem solving in the sense of manipulation of representations of choices in a search space. The goal-action-sensory system triple is highly evolved and integrated to carry out, in a goal-driven way, such action sequences.

When such action sequences are not immediately available for the goal, there are a number of options. Working memory may contain abstract representations of problem space alternatives. The problem space and the operators available may have not only abstract symbolic components, but imagistic components as well. Working memory may also contain previously developed sequences of solutions or pointers to external methods, algorithms, or models. Some of the subgoals are best accomplished by action sequences, some by operators that are specific to the image modality (e.g. *reasoning with mental images*), some by application of abstract knowledge operators, and some by invoking

external agents and models. Many of the subgoals can be accomplished just by interacting with the world or sensing the world rather than by reasoning on complex representations. A common way of avoiding complex reasoning is to leave representational markers in the physical world, and use action and sensory operators to 'read off' the information.

The above description emphasizes how much of real problem solving is dominated by the fact that the agent is situated in the world, and how artificial a pure symbolic representation manipulation view can be for many problems. At the same time, the above picture is admittedly schematic. A number of important issues remain unsolved. We already referred to the problem of the mechanisms by which knowledge in working memory is constructed in response to goals. How the sensor-action system is integrated with deliberation in an abstract sense requires many details to be worked out, but it sets a research agenda that is different from that of traditional AI.

Content-driven AI and microstructural accounts are both needed. In a strange way, the perspective we just outlined validates both traditional AI and the new emphasis on microstructure. Traditional AI, with its emphasis on knowledge and the distinctions needed to express it, has tried to wrestle content down. It has been able to do this pretty well up to a point, but because it is not embedded in a theory with appropriate microstructure and environmental interaction, ends up *over-idealizing* content and missing the form in which knowledge really emerges. The microstructural accounts have potential to explain the genesis and evolution of knowledge, and, to the extent that they base themselves on some aspects of biological neural systems, can explain aspects of continuity in cognition between higher animals and humans. It is also often hoped that the content problem in AI can be solved by AI systems that learn from scratch or with little initial knowledge. That is, the hope is that learning will obviate the need to develop knowledge level distinctions. That seems highly unlikely for reasons of complexity, both in time and in the environmental specification, but also due to the need for specifying appropriate initial states. It is more likely that the learning theories will give broad insights about content that might place useful constraints on knowledge level theories. Thus the content-driven AI picture and the microstructure-driven new architectural views need to work side by side for quite a while, hoping to meet in various ways and places for mutual benefit.

Hierarchy of leaky architectures and cognitive explanations. We have mentioned connectionism, dynamical systems, and Edelman's selection machine as three contending proposals for the subdeliberative architecture, and no doubt there will be many others over time. But to look for a 'correct' answer to the cognitive architecture may be to commit an error in reification, in believing that there exists one architecture that can be factored off the physical brain in such a way that the architecture corresponds to and only to cognition (or more generally mentality). In the introductory section on dimensions for thinking about thinking, we discussed the problems associated with factoring off a cognitive architecture from a mental architecture. A similar issue arises in the belief that a mental architecture can be factored off the physical brain or the body, and that a clearly defined set of functionalities can be identified to define mind. What we have in the brain is a biologically evolved complex piece of matter working at many levels, informational, chemical and electrical. Certainly different stances can be taken towards it for different analytical purposes, but believing that there exists a separable architecture called the mental, especially one that has a description at one level, may be Platonism run amok.

If this view is right, then we can see the contending proposals for the subdeliberative architecture as approximate descriptions, at somewhat different levels, of a physical reality called brain, which in turn is the basis for a host of behaviors that have a mentalistic description.

Consider the mathematical description of an economy in a human society. It would be strange to regard the economy as the reality which just happens to be implemented on humans. Description of an economic model is an approximate description of certain types of activities in human society. This is the analogy that we would like the reader to keep in mind as we describe our view of hierarchy of cognitive architecture descriptions.

In this view, the Edelman selection machine is a convenient and approximate description of a machine which is really a complex chemical machine. At a higher level, dynamical systems provide another approximate description, with connectionist descriptions providing yet another level of description. We think that when the selection machine organizes itself to perform some task, say perception, it should be possible to see in it a description of evidences being combined, the language in which connectionism works. At the top level we have the knowledge level description of the agent in terms of knowledge and goals. Each of these descriptions captures some aspects and functionalities, but misses others.

However, this picture of virtual machines all lined up vertically, the deliberative architecture on top of the recognition architecture on top of, say, a dynamical systems architecture which in turn is on top of

something else and so on all the way down to chemistry and physics, might give a false picture of perfect implementations of a higher level by a lower level. Biological brains do not really have cleanly lined up architectures in the way that computers do. In artifacts like computers, we as designers have conceptualized a pure information processing machine and have created a complete one-to-one correspondence between the elements of that and the elements of a physical machine. Except when the machine malfunctions we never have to worry about the lower level machine. In computer software design each level of architecture, each virtual machine, sits cleanly upon the one beneath it without the one beneath it showing through at all. Each level is smooth and closed and separate with respect to other levels of the architecture. This sort of architectural arrangement has guided much of our thinking about human cognitive architecture.

However, in a biologically evolved object like the human brain such a clean separation between levels of architecture and between software and hardware is impossible. This is because, first of all, these architectures we have been describing are all 'leaky' virtual machines. That is, when the surface structures are stressed, or under certain situations, the underlying machine shows through. There are layers of representational structures and representations from other layers peak through at any given layer. Like in the case of vision, where in certain optical illusions the physical structure of rods and cones shows through the interpretive architecture, the architecture of the underlying machines literally shows through in certain circumstances. The cognitive phenomena are thus not all going on at one level of architecture. Secondly, these layers of architectures are not complete, i.e. each level of description does not fully account for all the phenomena of interest. Given some complex mental activity, explanation of some aspects can be given by the Knowledge Level, for some we will need to appeal to the properties of the connectionist architecture, for some to the properties of the selection machine, and for others we may simply need to appeal to chemistry and other physical properties.

What description we use to account for the phenomena depends upon our goals. The cognitive phenomena we are looking at are not going to admit of any single level of explanation. They are very multi-dimensional, and for some purposes we can account for the behavior by referring to the deliberative machine, but for other purposes that will not do, and we will have to account for the behavior by reference to a lower level of the architecture. This means that the information processing architectures that we see underlying human cognitive behavior are architectures that we have abstracted for certain classes of purposes.

This is not to espouse a form of relativism, however. Not everything counts. There are lots of machines that could not be brought up as virtual machines by the brain. Interestingly, all the virtual machines that we considered, from Soar to connectionist systems to Edelman's path selection machines, have a special feature: they all are oriented towards adaptation and learning. Thus, there is a relationship between learnability and being capable of being a virtual machine of interest. There are facts of the matter to be investigated and discovered. We can ask of a proposed virtual machine, what work does it do? How is it useful as level of explanation? We can also ask of a particular task how is it being done? What sort of architecture is being used to accomplish it? Although we can potentially model each individual function of cognition, there may be no abstract platonic engine which accounts for all and only cognitive, or all and only mental, behavior. There may well be just various cognitive functions and various machines that can be used to explain those functions.

Concluding remarks

We started by asking how far intelligence or cognition can be separated from mental phenomena in general. We also suggested that the problem of an architecture for cognition is not really well-posed, since, depending upon what aspects of the behavior of biological agents are included in the functional specification, there can be different constraints on the architecture. That is, it is not clear that, from an architectural perspective, the idea of a cognitive architecture is a natural kind. Nevertheless, we said, we can talk about cognition as a coherent phenomenon of interest if we think of it as that behavior in which we ascribe knowledge states to the agent. Newell's Knowledge Level view of an agent is based on a similar point of view about a cognitive agent.

We reviewed a number of issues and proposals relevant to cognitive architectures. The computer metaphor has had its day, but, we argued, the information processing language has significant explanatory powers left. We ended with the position that the search for an architectural level that will explain all the interesting phenomena of cognition was likely to be futile. Not only are there many levels each explaining some aspect of cognition and mentality, but the levels interact even in relatively simple cognitive phenomena. Ultimately even physics will account for some mental phenomena.

By treating mentality, not to speak of its cognitive component, as ultimately not fully separable from the physical substrate, we are not being pessimistic about

the prospects for cognitive science and AI, just being careful about what one might expect. In one sense, this view reinforces the arguments for the need for grounding[27], and being and growing as real humans, as the ultimate requirement for achieving the kind of mentality that we have. On the other hand, explanations of all sorts of mental phenomena can come at various levels. We can build problem solvers, perceivers, cognizers and so on, and depending upon their physics they may have their own version of mentality. There is no need for AI or cognitive science to insist on the various Separability Hypotheses being true in all details for getting nearer and nearer to the goals of explanation and simulation of mind.

1. Searle, J. R., Minds, brains and programs. *Behav. Brain Sci.*, 1980, 3, 417–424.
2. Edelman, G. M., *Neural Darwinism: The Theory of Neuronal Group Selection*, Basic Books, New York, 1987.
3. Edelman, G. M., *The Remembered Present: A Biological Theory of Consciousness*, Basic Books, New York, 1989.
4. Newell, A., *Unified Theories of Cognition*, Harvard University Press, Cambridge, MA, 1990.
5. Dennett, D., *The Intentional Stance*, MIT Press/Bradford Books, Cambridge, MA, 1987.
6. Rumelhart, D. E., McClelland, J. L. and the PDP research group (eds.), *Parallel Distributed Processing: Essays in the Microstructure of Cognition, Foundations*, MIT Press/Bradford Books, Cambridge, MA, 1986.
7. Putnam, H., *Representation and Reality*, MIT Press/Bradford Books, Cambridge, MA, 1988.
8. Dreyfus, H., *What Computers Cannot Do: The Limits of Artificial Intelligence*, Harper and Row, New York, 1972.
9. Schank, R. C., *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*, Cambridge University Press, New York, 1982.
10. McCarthy, J. and Hayes, P. J., Some philosophical problems from the standpoint of artificial intelligence, *Machine Intell.*, 1969, 6, 133–153.
11. McCarthy, J., Circumscription: A form of non-monotonic reasoning, *Artif. Intell.*, 1980, 13, 1–2, 27–41.
12. Newell, A. and Simon, H., *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, NJ, 1972.
13. Patten, T., Geis, M. and Becker, B., Toward a theory of compilation for natural language generation, *Comput. Intell.*, 1992, 8(1), 77–110.
14. Chandrasekaran, B., Roles of logic in Artificial Intelligence, *Vivek: A Quarterly in Artificial Intelligence*, National Centre for Software Technology, Bombay, 1991, 4(2), 13–15.
15. Chandrasekaran, B., Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design, *IEEE Expert*, 1986, 1(3), Fall, pp. 23–30.
16. Minsky, M., A framework for representing knowledge, *The Psychology of Computer Vision* (ed. Winston, P. H.), McGraw Hill, New York, 1975, pp. 211–280.
17. Minsky, M., *The Society of Mind*, Simon and Schuster, New York, 1986.
18. Fodor, J. A. and Pylyshyn, Z. W., Connectionism and cognitive architecture: A critical analysis, *Cognition*, 1988, 28, 3–71.
19. Shastri, L., Connectionism and the computational effectiveness of reasoning, *Theor. Ling.*, 1990, 16(1), 65–87.
20. Pollack, J. B., Recursive distributed representations, *Artif. Intell.*, 1990, 46(1), 77–105.
21. Pollack, J. B., Review of Unified Theories of Cognition, in *Artif. Intell.*, 1993, in press.
22. Skarda, C. A. and Freeman, W. J., How brains make chaos in order to make sense of the world, *Behav. Brain Sci.*, 1987, 10, 161–195.
23. Crutchfield, J. P. and Young, K., Computation at the onset of chaos, in *Computation, Entropy and the Physics of Information* (ed. Zurek, W.), Addison-Wesley, Reading, MA, 1989.
24. Fodor, J. A., *The Modularity of Mind: An Essay on Faculty Psychology*, MIT Press/Bradford Books, Cambridge, MA, 1983.
25. Clancey, W. J. and Roschelle, J., Situated cognition: How representations are created and given meaning, Technical report, Institute for Research on Learning, Palo Alto, CA 94304, USA, 1991.
26. Iran-Nejad, A., The schema: A long-term memory structure or transient functional pattern, in *Understanding Readers' Understanding: Theory and Practice* (eds. Tierney, J. *et al.*), 1980, Lawrence Erlbaum, Hillsdale, 1987.
27. Harnad, S., The symbol grounding problem, *Physica*, 1990, D42, 335–3466.

ACKNOWLEDGEMENTS. B. Chandrasekaran's work in the preparation of this paper was supported by US Defense Advance Research Projects Agency via contract F-49620-89-C-0110, monitored by Air Force Office of Scientific Research. We thank Tom Bylander, John Josephson and Jordan Pollack for their comments on a draft of this paper, and Prof. Narasimhan for the invitation to write this article.

Chapter 4

Perceptual Representation and Reasoning

B. Chandrasekaran and N. Hari Narayanan

Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University, Columbus, Ohio 43210
email: chandra@cis.ohio-state.edu, narayan@cis.ohio-state.edu

Abstract

A common view of reasoning in cognitive science is that it is a process that operates on abstract sentential representations. This view implies a separation of reasoning from sensory perception. Consequently, the study of perception has proceeded relatively independently of the study of various reasoning strategies that humans employ. In this paper we argue that there are many commonsense situations in which human reasoning is tightly coupled with perception, in particular with perceptually represented experiential knowledge. This type of reasoning is referred to as perceptual reasoning. We explain perceptual reasoning in terms of experientially acquired perceptual inference rules, and briefly discuss how this relates to a previous proposal about representations that underlie visual perception and imagery [5]. Finally, the implications of this stance are discussed.

4.1 Introduction

Professor Yoh-Han Pao's research interests have spanned a wide variety of issues in Artificial Intelligence. The subject of this paper, we think, reflects at least some of the range of Prof. Pao's interests: pattern recognition, vision and problem solving. We are pleased to discuss in this paper the close integration between perception and problem solving that exists in people and make some proposals about how computers can be designed which take advantage of diagrams and images in problem solving.

Sensation, perception, cognition, and reasoning have all been subjects of study by cognitive psychologists. How are these related? One answer, from an information

This paper is a revision of a talk presented at the AAAI Spring Symposium on Reasoning with Diagrammatic Representations, Stanford University, Palo Alto, California, USA, March 25-27, 1992. An earlier version appears in the Working Notes of the symposium, pp. 24-29.

Intelligent Systems, Edited by L.S. Sterling
Plenum Press, New York, 1993

processing perspective, is that reasoning operates on information about the world around us. This information is made available to the cognitive system through the processes of sensation, perception, and cognition. Are these four processes strictly sequential, influencing each other in a unidirectional way? While it seems obvious that sensation must precede perception, it is not evident that cognition must strictly follow perception or that reasoning must operate solely on post-cognitive representations. The work of Biederman [4] on human object recognition, for example, suggests that perception of a few shapes in an object can trigger recognition which then can influence perception. Deliberative reasoning has been typically characterized as operating on post-cognitive representations and therefore divorced from acts of perception that produced them. However, the view that human reasoning and behavior are tightly coupled with (situated in) perception is gaining increasing credence. An excellent illustration of perceptually grounded reasoning is given in Shrager's account of commonsense perception [19]. Another confounding phenomenon in the visual modality has been that of mental imagery [20]. Any theory of visual perception and cognition that postulates some underlying representations and mechanisms has to account for this phenomenon.

Humans are adept at making plausible inferences in situations in which the reasoning employed typically involves perception, cognition, and imagination. Ideas emerging from recent research (such as those cited above), namely, that perception and cognition influence each other, that any theory of perceptual representations and mechanisms should account for imagery also, and that reasoning is sometimes tightly coupled with perception, are very relevant to analyzing commonsense reasoning in such situations. In this paper we take a careful look at specific scenario that involves reasoning about *perceptual* and *imaginal* events. This is called *perceptual reasoning*. It is argued that perceptual reasoning can be characterized in terms of experientially acquired perceptual inference rules. Then we briefly describe some properties of perceptual representation and architecture. It is suggested that the specialized (to the visual modality) nature of perceptual architecture and modality-specific operations on perceptual representations that it provides together can account for mental imagery and internal visualizations during perceptual reasoning. What is outlined here are the beginnings of a theory of commonsense visual reasoning based on the twin ideas of modality-specific representations and inference rules with perceptual and conceptual content [6]. Finally, the implications of our stance are discussed.

4.2 Perceptual Reasoning

4.2.1 A Scenario

Consider the following situation. You are seated with others around a table and you notice that someone sitting beside you is about to throw a rock. You then notice that the rock, if thrown, will hit a glass-paned window outside which a child is playing. Your immediate reaction will be to restrain the potential rock thrower, in order to prevent the child from being hurt.

4.2.2 Analysis

Let us now analyze the reasoning process behind this prediction that the child is likely to get hurt. This inference may be seen to be a direct consequence of another inference; that the rock will shatter the glass pane resulting in an outwardly spreading spray of glass shards that will hit the child. This inference is preceded by yet another inference about the possible trajectory of the rock. The first inference in this chain, namely that the flying rock will hit the glass pane of the window, is derived by the visual system using perceptual and motor operations (in this case scanning, that may involve a mere shift of attention, eye movements, or even turning the head depending on the distances involved, would provide the needed geometric information) in response to the internal goal of predicting the possible trajectory of the rock. The second inference, that the glass will shatter resulting in an outwardly spreading shower of shards, is more interesting. It is clearly not derivable from the environment since it has not happened yet. However, we possess experientially acquired knowledge that glass panes shatter when hit by flying objects. While this "chunk" of knowledge is verbalizable as above, it consists of more than what this verbalization expresses. This is evident from the fact that we are capable of distinguishing situations that "look" alike. For instance, we do not always predict shattering when seeing or thinking about a flying object about to hit a transparent pane. If we also know that the pane is made of non-breakable plastic or that the object is made of soft rubber, we will not predict that the pane will shatter. Note that the relevant knowledge about the material properties of the pane and the object is conceptual (non-visual) in nature. Thus the knowledge that is brought to bear for making a prediction has a conceptual component. It has a perceptual component as well, which is what facilitated the internal visualization of the rock hitting the pane, the pane shattering, and the shards flying outwards in the general direction of the rock's trajectory. This visualization utilizes representations supplied by perception and imaginal operations on them provided by the perceptual architecture. It is concluded from this visualization that the shards may hit the child. At this point the knowledge that a child hit by flying objects may be hurt kicks in (this knowledge may be purely conceptual, or for very imaginative people, may have a perceptual component that allows them to visualize glass shards penetrating the skin), generating the inference that the particular child currently playing outside the window is likely to be hurt. This of course, is the motivation behind the restraining act.

This scenario brings out some very interesting aspects of commonsense reasoning. One is that quantitative information, such as velocity of the rock on impact, does not seem to have been used by the reasoning process. Therefore this type of reasoning may be classified as qualitative reasoning. Other interesting aspects become evident when one considers what kinds of mental or physical operations were employed in order to make the three inferences and to which entities these operations were applied. The first inference about the rock hitting the window pane was made by scanning the scene along a predicted trajectory. Thus, this reasoning involved perceptual and motor operations applied to the environment (eye movement for scanning) as well as computations on an internal representation of the environment (predicting a trajectory). The second inference about the effects of a collision between the rock and the window pane was done by an internal visualization guided

by our experiential knowledge. Thus, this reasoning involved imaginal operations applied to an internal representation of the environment. The third inference about the child being hurt was made by applying conceptual knowledge to information derived from the visualization.

It is clear that reasoning in this scenario is not merely a process of manipulating sentential or propositional knowledge. Rather, it is a process of goal-directed inferences made from the environment and its internal representation perceptually and imaginally. Generating predictions during such reasoning is mediated by experiential knowledge that we have about events in the physical world. This is the phenomenon that we call perceptual reasoning.

4.2.3 Perceptual Rules

An interesting question is how event predictions, which are experienced as vivid internal visualizations, are made during perceptual reasoning. We postulate that these predictions are driven by perceptual inference rules that are acquired from the experience of interacting with the physical world. A perceptual rule is a piece of inferential knowledge whose antecedent and consequent have perceptual components. These components are *abstract* representations of perceptual events, which can be matched to a large number of particular situations. In other words, a perceptual rule is essentially a learned association between two perceptual events in a sufficiently abstract form that it will match a class of particular perceptual events. Such perceptual rules may often be verbalized, but such verbalizations typically *close* some of the information contained in the original perceptual version of the rules. For example, the content of a perceptual rule relevant to the aforementioned glass shattering scenario may be verbalized as "if a flying object hits a glass pane, the pane is likely to shatter," but the corresponding internal representation has perceptual components that preserve spatial properties which can be used directly for prediction. The acquisition of particular antecedent-consequent perceptual event associations and their generalization into abstract perceptual rules are the result of learning from experience.

A rule must also have conceptual (non-visual) conditions that allow discrimination among perceptually similar situations, to some of which the rule is applicable while to others it is not. For example, one conceptual condition of the rule on shattering could be that the object be hard and heavy. While perceiving or visualizing a collision between an object and a transparent pane, determining whether the object is indeed hard and heavy requires extraneous (to perception) knowledge. We may know that rocks are generally hard and heavy and that the flying object in the current situation is a rock, and thereby conclude that this conceptual condition is indeed satisfied. Thus, the perceptual and conceptual components together serve to determine the rule's applicability to any particular situation.

The antecedent of a perceptual rule contains abstract specification of a perceptual event that will match with a class of particular perceived or imagined events. For instance, the abstract specification of the collision event in the antecedent of the shattering rule would match with a variety of perceived or imagined collisions between different objects and glass panes of different shapes and orientations. This raises the question of how a perceptual event is specified abstractly. With per-

ceptual representations that are hierarchically structured with levels of increasing detail (such as the object-centered representations of Marr and Nishihara [12] this is possible since the antecedent specification can be such that it will match the top level(s) of perceptual representations of events belonging to a particular class (e.g., the class of collisions between objects and panes). This match need not be affected by details of shapes and orientations of objects and panes involved since these will be represented at lower levels of the representation hierarchy. The consequent of a rule contains an operational specification of a predicted event, which the perceptual architecture uses to modify the particular representation that matched the rule's antecedent so as to reflect the predicted event's occurrence. This modification of the perceptual representation drives the internal visualization of the predicted event. Thus we are able to visualize the glass pane shatter, for example, immediately following an imagined impact of a rock flying through air.

Thus, perceptual rules have antecedents consisting of abstract specifications of perceptual events which can match with a variety of particular perceptual events (actually perceptual representations of events delivered by perception), and conceptual conditions that provide a finer discrimination capability. Their consequents contain plausible predictions. Such rules facilitate not only the making of predictions, but also the visualization of these predictions. How are such rules used? Particular perceptual events, whether witnessed or imagined, that match the abstract specifications of a rule will activate it. If the corresponding conceptual conditions are also met, the rule is applied, resulting in the generation of an inference (prediction) and the modification of the perceptual representation of the triggering event to reflect the effect of the predicted event. This enables one to visualize the predicted event. All this takes place within the perceptual architecture.

This conception of perceptual reasoning is somewhat different from the traditional view of reasoning and deliberation. It was partly motivated by viewing perception and imagery from the artificial intelligence perspective of reasoning. The aim of all this theorizing has been to put forth one explanation of a kind of reasoning that humans engage in routinely. This sort of reasoning has not hitherto received much attention in artificial intelligence or cognitive psychology.

4.2.4 Perceptual Representations

In the foregoing description we referred more than once to hierarchically structured perceptual representations and a perceptual architecture. Therefore, a discussion of what we mean by these terms is in order. The representation of visual information and its relation to the phenomenon of mental imagery have attracted a great deal of attention from cognitive psychologists. There has been considerable debate [1, 10, 15, 20] about postulating analog representations for imagery as opposed to uniform propositional representations. While all contentious issues regarding imagery may not yet be fully resolved, some properties of perceptual representations can be gleaned from the empirical and philosophical literature on this topic. Four such properties are the following.

(1) Perceptual representations must be compositional, since we can compose different (even previously unseen) mental images quite easily and rapidly. Bieder-

man's [4] theory of recognition by components suggests that such representations must be componential, and hence compositional.

(2) Perceptual representations must contain information about the represented object at different levels of detail since we are able to imagine a previously seen object at different resolutions. This suggests that the representations may be hierarchically structured with levels of increasing detail, along the lines suggested by Marr and Nishihara [12].

(3) Perceptual representations must support an "internal depiction" of the represented object since a mental image is an experience of such a depiction. Indeed, researchers have postulated mechanisms such as the visual buffer [10] and symbol-filled arrays [20] to account for this depictive property.

(4) Perceptual representations must also have a descriptive component encoding structural information, as Hinton [7] points out.

We have previously argued [5] that hierarchically structured descriptive representations, when loaded from long term memory into a specialized (to the visual modality) architecture that provides modality-specific operations on these representations, can give rise to the experience of mental imagery. Furthermore, we believe that these perceptual representations are similar, in a sense, to the discrete symbolic representations used by computers. A discrete symbolic representation consists of structures of symbols composed according to well-defined rules of formation. A symbol is just a token, and by itself devoid of any meaning. Therefore the distinctive property of such a representation is that its semantics derives from its interpretation, by means of operators provided on it, by the architecture of the system in which it resides. We suggest that perceptual representations are similar to discrete symbolic representations in that they also have this property. In other words, the experience of mental imagery and of applying operations like scanning on mental images arise not from any inherent analog property of the representations themselves, but from an interpretation of perceptual representations by a modality-specific architecture.

This characterization of perceptual representations is different from an analog or propositional characterization. On one hand, unlike analog representations, these are not depictive, but are descriptive. On the other hand, unlike propositional representations, their interpretation is not based on some universal truth semantics, but is dependent on the architecture in which they reside. Since in this view the meaning of a perceptual representation derives from its interpretation by the underlying architecture and processes that operate upon it, properties exhibited by such a representation are not intrinsic to the representation itself, but stem from the modality-specific architecture that supports it and processes that operate on it. This feature provides a way to explain how non-analog representations can, in principle, give rise to the experience of imagery by virtue of being interpreted by an underlying modality-specific architecture.

Perceptual representations are hierarchical structures comprising descriptors of visual attributes such as shape, color, texture, etc., and of spatial relations among elements. The hierarchical structure reflects different levels of detail at which a perceived scene is encoded. The depth of the hierarchy and resolution of description at different levels depend upon aspects (such as objects in the scene that were

attended to, how closely they were looked at, etc.) of the act of perception which produced the perceptual representation.

A collection of descriptors in a perceptual representation that together correspond to a distinct element of a scene (such as an object or part of an object) forms a percept. A perceptual representation is made up of multiple percepts. A percept is a basic unit or building block of perceptual representations. It describes all visually perceived information about a distinct element of an image. Each percept has a corresponding mental image that results from its interpretation. This image in the mind's eye is the depictive counterpart of the percept, which is descriptive in nature. The description and the depiction are two sides of the same coin. Thus a perceptual representation consisting of percepts is both an internal description of a perceived scene and a recipe for the composition of a corresponding mental image through interpretation. Percepts provide compositionality, i.e., allow one to compose perceptual representations (and corresponding mental images) from percepts that are parts of representations of different images.

A mental image results from accessing relevant percepts in memory, bringing these into the perceptual architecture, and composing these appropriately. This visual-modality specific architecture provides imaginal operations on the representation. Just as interpretation generates an experience of mental imagery, the invocation of an imaginal operation (e.g., scanning) concomitantly creates the corresponding experience (e.g., that of scanning an image with the mind's eye). The imaginal operations that the perceptual architecture provides on perceptual representations are similar to the operations that the visual system employs under perceptual conditions, but they may not be identical [17].

Note that though the analogy of discrete symbolic representations interpreted by an underlying computational architecture was used to explain perceptual representations, these need not necessarily be symbolic in nature. These may instead be realized as patterns of weights or strengths in a neural substrate. The preservation of aforementioned properties is what is important. Our proposal is also neutral with respect to the postulation of an analog medium for mental images and internal visualizations. It may very well be that interpretation of the perceptual representation of a scene results in the creation of a surface display in a visual buffer as Kosslyn [10] suggests or the creation of a symbol-filled array as Tye [20] suggests. On the other hand it may be the case that this interpretation merely produces the same pattern of neural activation as that created by the topographic projection of the retinal image into the visual cortex during perception, thus creating the experience of mental imagery. The main point here is that representations that are neither analog nor propositional and an underlying perceptual architecture that provides operations specific to the visual modality can possibly explain the phenomenon of imagery as well as support perceptual reasoning.

4.3 Discussion

In this paper a phenomenon termed perceptual reasoning was illustrated with an example and explained in terms of perceptual inference rules. Subsequently, we described properties of perceptual representations and architecture that may

underlie visual perception and imagery as well as support perceptual reasoning. For the moment at least, the strength of ideas in this paper lies in the richness of their implications rather than on the extent of their empirical support.

One point of this paper, therefore, is to suggest that it is worth conducting experiments to gather (supporting or opposing) direct evidence. There is some indirect evidence available. Yates and colleagues [21] report an experiment which showed that individuals solving motion prediction problems, such as predicting the path of a ball released from the end of a rotating sling, subjectively experienced reported solutions as the result of a mental enactment of the problem situation. Furthermore, they suggest that the source of this enactment appeared to be a number of relatively unsystematic, mutually inconsistent, and situation-specific prototypes based on experience, which capture typical aspects of motion. A precise characterization of perceptual inference rules in terms of content and organization, if indeed these are psychologically real, requires much empirical research of the sort undertaken by Yates and colleagues. The phenomenon of "perceptual fluency" observed by Jacoby and Dallas [8] is also relevant. They found that following a 24-hour delay after subjects studied a word-list, their conscious recognition of these words was at near-chance levels. But the subjects were twice as likely to recognize these words, compared to control words, in a tachistoscopic-recognition paradigm. It is argued that prior exposure leads to "perceptual fluency". On a similar vein, Proffitt and Keiser [14] found that subjects were very good at rejecting anomalous motions when shown videotapes of actual and simulated motions. If prior exposure can lead to perceptual fluency in recognition and classification, is it possible that it can also lead to perceptual fluency in reasoning? Koedinger and Anderson [9] show that expert problem solving behavior in geometry can be modeled in terms of the instantiation of schemas which organize problem solving knowledge around prototypical images. The idea of perceptual inference rules in which perceptual events cue predictions is not too far removed from this. Anderson [2] has pointed out that production systems consisting of rules with antecedents and consequents are not incompatible with imagery processes.

What about the implications of this proposal? One implication is that the true nature of representations underlying perception and imagery may be closer to a middle ground between the extreme positions within the propositional and analog schools of thought. More relevant to the focus of this paper, however, is the notion that reasoning may be situated or grounded in perception. Another implication, therefore, is to suggest that much of commonsense reasoning is tightly coupled with perception and that perceptual representations, not just conceptual knowledge, play a crucial part in the reasoning process. Just as the paradigm of case-based reasoning [18] provided an impetus for computational investigations of memory-based reasoning, our hope is that proposals like this will provide an impetus for computational studies of perception-based reasoning. In fact, there is already a move toward legitimating the formal study of the use of diagrams and pictures in reasoning [3]. Furthermore, studies of human reasoning also ought to pay closer attention to reasoning processes that are closely coupled with perception and perceptual representations. A broader implication is that reasoning may be coupled with perception in modalities other than vision as well. We support a previously

expressed view [11, 17] that research on reasoning in artificial intelligence should explore connections to vision and imagery.

Our current research is on developing a computational model of perceptual representation and reasoning in the domain of spatially interacting objects depicted in diagrams. Limited space precludes a detailed discussion here (see [13] for an overview). However, two aspects of this computer system that directly relate to the focus of this paper and therefore worthy of mention are the computational realizations of perceptual representation and inference rules. The computer representation of an object configuration diagram consists of a hierarchy of symbolic descriptors and bitmap-based depictions, both of which are accessible to reasoning procedures. This representation is compositional, hierarchical, partly depictive, and partly descriptive. Representational devices called *visual cases* are used to encode the predictive knowledge of perceptual inference rules. A visual case consists of visual cues, conceptual conditions, and predicted events. We have developed a suite of over sixty cases for the current domain (see fig. 4.1 for an example). Given a prediction problem that consists of a multi-object configuration diagram and the initial motion of one object, the system will go through cycles of visual analysis and deliberation, and predict the temporal evolution of the configuration. Visual cases come into play during deliberation. Visual cues are used for matching cases to the current configuration, conceptual conditions provide additional discrimination capability, and predictions of cases found to be applicable drive the next cycle of reasoning. It is worth noting here that visual cues make visual case matching similar to the instantiation of diagram configurations [9], and conceptual conditions provide a means for the computational modeling of cognitive penetrability [16].

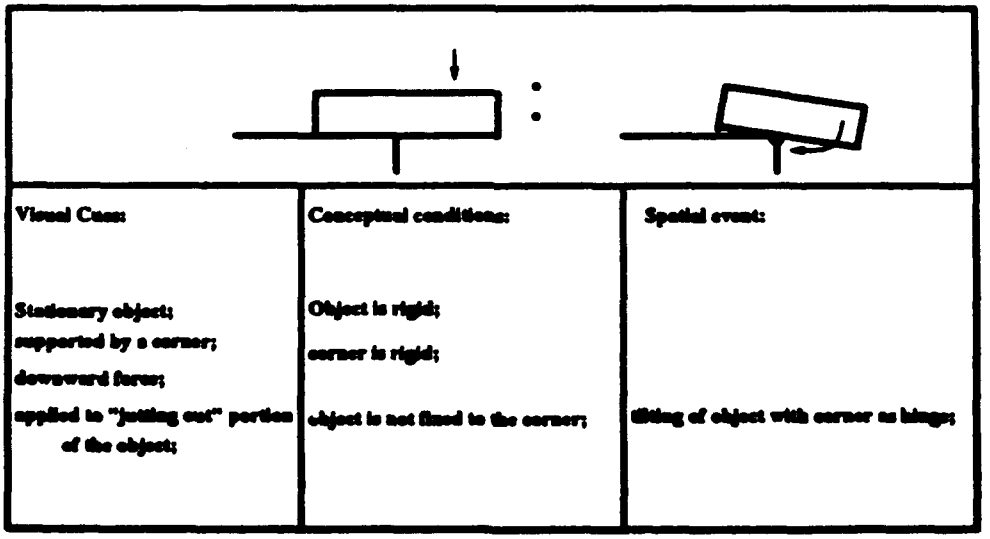


Figure 4.1: An Example of a Visual Case

Acknowledgments

This research has been supported by DARPA under AFOSR contract F-49620-89-C-0110 and a BP fellowship to the second author.

References

[1] Anderson, J. R. (1978). Arguments concerning representations for mental images. *Psychological Review*, 85, 249-277.

[2] Anderson, J. R. (1983). *The Architecture of Cognition*. Harvard University Press.

[3] Barwise, J. & Etchemendy, J. (1990). Valid inference and visual representation. In Zimmerman & Cunningham, (Eds.), *Visualization in Mathematics*, Mathematical Association of America.

[4] Biederman, I. (1987). Recognition-by-components: a theory of human image understanding. *Psychological Review*, 94, 115-147.

[5] Chandrasekaran, B. & Narayanan, N. H. (1990a). Integrating imagery and visual representations. *Proceedings of the 12th Annual Conference of the Cognitive Science Society*, 670-677.

[6] Chandrasekaran, B. & Narayanan, N. H. (1990b). Towards a theory of commonsense visual reasoning. *Lecture Notes in Computer Science*, 472, Springer-Verlag, 388-409.

[7] Hinton, G. (1979). Some demonstrations of the effects of structural descriptions in mental imagery. *Cognitive Science*, 3, 231-250.

[8] Jacoby, L. & Dallas, M. (1981). On the relationship between autobiographical memory and perceptual learning. *Journal of Experimental Psychology: General*, 3, 306-340.

[9] Koedinger, K. R. & Anderson, J. R. (1990). Abstract planning and perceptual chunks: elements of expertise in geometry. *Cognitive Science*, 14, 511-550.

[10] Kosslyn, S. M. (1981). The medium and the message in mental imagery: a theory. *Psychological Review*, 88, 46-66.

[11] Lindsay, R. K. (1989). Qualitative geometric reasoning. *Proceedings of the 11th Annual Conference of the Cognitive Science Society*, 418-425.

[12] Marr, D. & Nishihara, H. K. (1978). Representation and recognition of the spatial organization of three dimensional shapes. *Proceedings of the Royal Society*, 200, 269-294.

[13] Narayanan, N. H. & Chandrasekaran, B. (1991). Reasoning visually about spatial interactions. *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 360-365.

[14] Proffitt, D. & Kaiser, M. (1986). Why you cannot see what holds a gyroscope up. Paper presented at the *Annual Meeting of the Psychonomic Society*.

[15] Pylyshyn, Z. W. (1981). The imagery debate: analogue media versus tacit knowledge. *Psychological Review*, 88, 16-45.

[16] Pylyshyn, Z. W. (1984). *Computation and Cognition: Toward a Foundation for Cognitive Science*. MIT Press.

[17] Reisberg, D. (1987). External representations and the advantages of externalizing one's thoughts. *Proceedings of the 9th Annual Conference of the Cognitive Science Society*, 281-293.

[18] Schank, R. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge University Press.

[19] Shrager, J. (1990). Commonsense perception and the psychology of theory formation. In J. Shrager & P. Langley, (Eds.), *Computational Models of Scientific Discovery and Theory Formation*, Morgan Kaufmann Publishers.

[20] Tye, M. (1991). *The Imagery Debate*. MIT Press.

[21] Yates, J. *et al.* (1988). Are conceptions of motion based on a naive theory or on prototypes? *Cognition*, 29, 251-275.

Reasoning Visually about Spatial Interactions

N. Hari Narayanan and B. Chandrasekaran
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210
U. S. A.

Abstract

This paper is concerned with how diagrams can be used for reasoning about spatial interactions of objects. We describe a computational approach that emulates the human capability of predicting interactions of simple objects depicted in two dimensional diagrams. Three core aspects of this approach are a visual representation scheme that has symbolic and imaginal parts, the use of visual processes to manipulate the imaginal part and to extract spatial information, and visual cases that encode experiential knowledge and play a central role in the generation of spatial inferences. These aspects are described and the approach is illustrated with an example. Then we show that reasoning with images is an emerging and promising area of investigation by discussing computational and cognitive research on imagery.

1 Introduction

Humans quite often make use of spatial information implicit in diagrams to make inferences. For example, anyone familiar with the operation of gears will be able to solve the problem posed in Figure 1 by imagining the rotary motion of gear1 being transmitted to the rod through gear2, resulting in the horizontal translation of the rod until it hits the wall. In such situations humans reason about spatial interactions not only by using conceptual knowledge, but also by extracting constraints on such interactions from a perceived image. This integrated use of visual knowledge (about spatial configurations) from the diagram and conceptual knowledge (such as the rigidity or plasticity of objects involved) is a very interesting phenomenon. In this paper we illustrate a computational approach that emulates this capability for solving simple motion prediction problems.

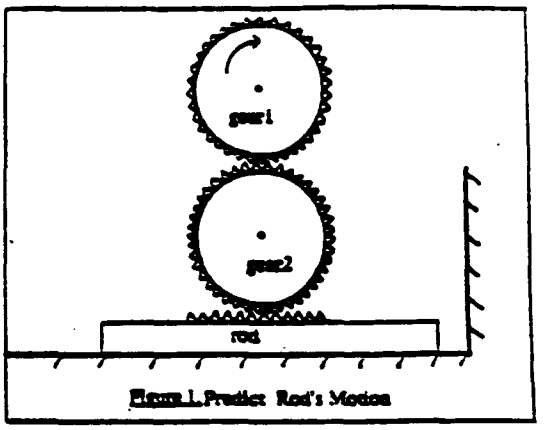


Figure 1. Predict Rod's Motion

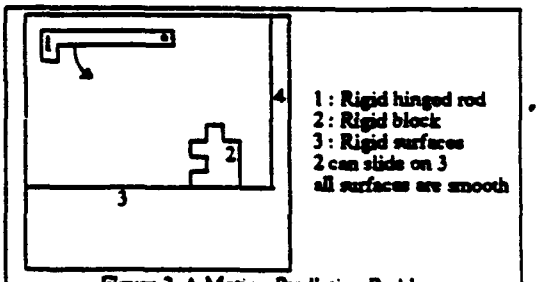


Figure 2. A Motion Prediction Problem

2 The Approach

2.1 Motion Prediction Problems

The class of problems we address is the following: given a two dimensional diagram of the spatial configuration of a set of objects, one or more initial motions of objects and relevant conceptual information about them, predict the subsequent dynamics of the configuration. Figure 2 shows a typical example.

2.2 Cognitive Inspiration

There is considerable evidence in cognitive science for the use of mental images by people when solving problems [Kosslyn, 1981]. Furthermore, introspective reports of people when given a diagram

To appear in *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, August 1991, Mountain View, CA: Morgan Kaufmann Publishers.

like Figures 1 or 2 and asked to predict motions indicated that by looking at the diagram they were able to visualize the motion of one object causing that of another through physical contact. They appeared to be using (the image of) the diagram in front of them directly to simulate motions in their minds. These reports indicated the following.

(1) Given a diagram depicting the problem, humans quite rapidly focus on localities of potential interactions.

(2) People also seem to simulate or project the motion to determine the nature of interactions that will occur.

(3) For reasoning about the dynamics (e.g., how will motion be transmitted after a collision?) humans bring conceptual knowledge (e.g., gears are rigid objects) and experiential knowledge (e.g., if an object collides with another, it typically transmits motion in the same direction) to bear on the problem.

We have developed an approach that emulates these capabilities.

2.3 Representation

The specification of a motion prediction problem consists of a scene depicting the spatial configuration of the objects involved and conceptual information about their properties (see Figure 2). The spatial configuration is represented using a "visual representation" while conceptual information about object properties is represented declaratively and linked with corresponding object descriptions in the visual representation. In our computer model the visual representation of a problem specification is interactively constructed prior to problem solving, whereas in the case of humans perceptual processes deliver such representations.

Mental representation of visual information and its relation to the phenomenon of mental imagery have been the foci of considerable research in cognitive science [Biederman, 1990; Finke, 1989; Kosslyn, 1981; Pylyshyn, 1981]. A central issue here is the question of how mental imagery that appears to be analogic in nature can arise from underlying representations that are considered to be propositional. One hypothesis regarding this issue [Chandrasekaran and Narayanan, 1990] is that representations for different sensory modalities are operated upon by interpreters that provide privileged operations specific to that modality, including binding the symbols in the representation to perceptual primitives in the corresponding sensory domain. Thus our answer to the above question is that symbolic representations of visual information are interpreted by mechanisms that are specialized to the visual modality and which provide operations tailored to this modality. These operations construct mental images using perceptual primitives in the visual domain. Visual representations in our computer model are therefore structured as multi-level hierarchies that contain *imaginal* descriptions and *symbolic* descriptions of the object configuration. Each level of the hierarchy contains a symbolic description and an imaginal description of the configuration at a certain resolution. The symbolic description is built from parametrized shape primitives like circles, rectangles, etc., whereas the imaginal description is a two dimensional pixel array of fixed width and height in which a configuration is depicted by object boundaries and is implemented as a bitmap. In the rest of the paper we will use the term "diagram" to refer to this boundary-based rendering of the object configuration. Thus the visual representation is dual (symbolic and imaginal) in nature. The two types of mental representations (surface images and deep encodings) that Kosslyn [1981] proposes reflect a similar duality.

The most interesting property of this representation is that it simultaneously provides abstract symbolic descriptions of an object configuration and directly captures, in the imaginal descriptions, specific spatial information about the object configuration (the extent of contact between two surfaces, for example). The justification for our decision to structure the symbolic descriptions in terms of parametrized shape primitives stems from shape representation theories that utilize primitives like geons [Biederman, 1990] and generalized cones [Marr and Nishihara, 1976]. Multiple levels of description are provided in the representation to allow visual processes to operate at different levels of resolution.

2.4 Reasoning

The basic model of reasoning is as follows. The system goes through a sequence of deliberative states. This sequence corresponds to the changes that the initial object configuration undergoes due to motion and interaction of objects. Each deliberative state represents a particular configuration that the objects assume at some point during this evolution of behavior. What distinguishes a deliberative state from other states is that it represents a configuration in which an interaction (such as collision) has occurred that will change the subsequent behavior of objects. The term deliberative refers to the necessity of "deliberation" that arises at these states in order to predict future behavior of objects. A significant characteristic of this deliberation is the combined use of perceptual information from the diagram and conceptual knowledge relevant to the situation.

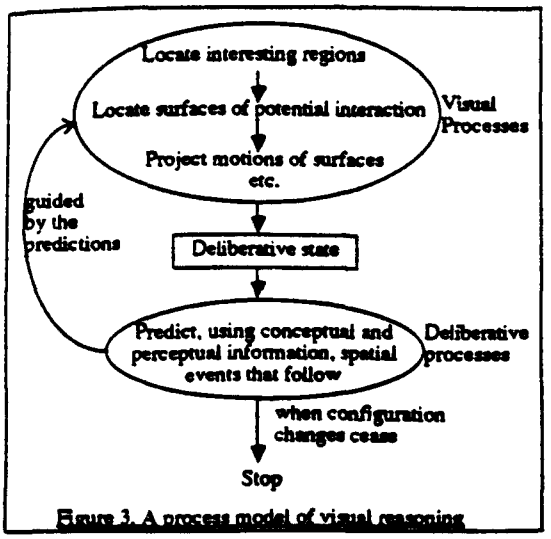


Figure 3. A process model of visual reasoning

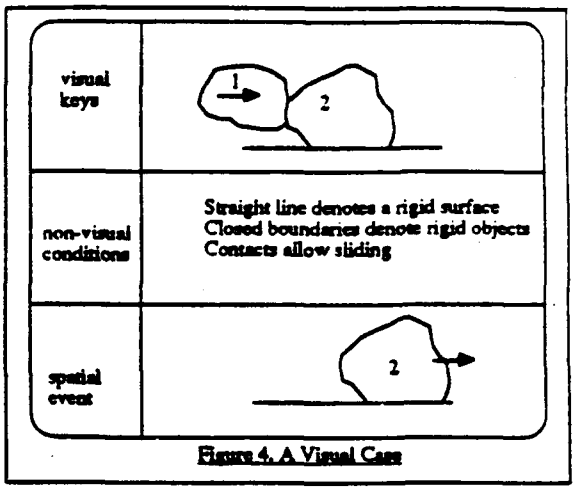


Figure 4. A Visual Case

The transition between deliberative states is accomplished by two groups of processes. In one, purely visual operations such as attention-focussing, scanning, boundary-tracking, and contact-detection are used to identify significant aspects of the current object configuration from the diagram (e.g., locating interesting regions, identifying surfaces of potential interaction, etc.), to reason about how the configuration will evolve (e.g., project surface motions), and to detect the next deliberative state. A deliberative state is detected by watching out for certain events as the configuration depicted in the diagram changes. The establishment of a new contact between objects, the elimination of a previously existing contact, the establishment of a new support relationship between objects, and the removal of a support relationship are some examples of events which indicate a deliberative state. Motivations behind and justifications for these processes derive from the extensive literature on mental imagery, some of which are discussed in section 3, and the work of Chapman [1990] and Ullman [1985] on visual routines. This group of processes corresponds to the human cognitive process of "imagining".

The second group of processes accomplish the aforementioned task of deliberation. Here knowledge about how interacting physical objects tend to behave under various conditions is used to predict the behavior of objects following the current deliberative state. We take a specific position on the form in which this knowledge is available and the way in which it is utilized. This is described next. A process model of visual reasoning is shown in Figure 3.

We believe that the knowledge humans bring to bear on making spatial inferences in similar situations is mostly acquired through experience, and so in the computer model experiential knowledge has been given a central role in deciding how to proceed from a deliberative state. Experiential memory is considered to be an organized and indexed collection of cases [Schank, 1982] and case-based reasoning is a computational paradigm for modelling the role of experience in problem solving [Kolodner and Simpson, 1989]. Therefore, representational structures called "visual cases" have been developed to encode knowledge applicable at deliberative states and to facilitate inferencing. Each case represents a typical spatial event. Since cases represent experiential knowledge, they may not be logically parsimonious or mutually exclusive. A visual case has three parts. One is information about spatial configurations to which the case is applicable. Cases are called "visual" because this information is visual in nature and is the "key" by which relevant cases get selected during reasoning. It may also be viewed as an "abstract" image that depicts the essential aspects of configurations to which the case is applicable. Because of this abstractness a case can be matched with a variety of specific configurations. This property obviates the need for a large number of cases. The second part is non-visual information that qualifies the visual part further and it is used for deciding the applicability of a case to a particular situation. The third part is a predicted event affecting objects in the spatial configuration represented by the case. This event may specify a state change (e.g., a directional force being applied on an object), a continuous change (e.g., an object moving in a particular direction), etc.

Humans are skilled at blending perceptual and conceptual information in generating spatial inferences. To illustrate this, first consider your

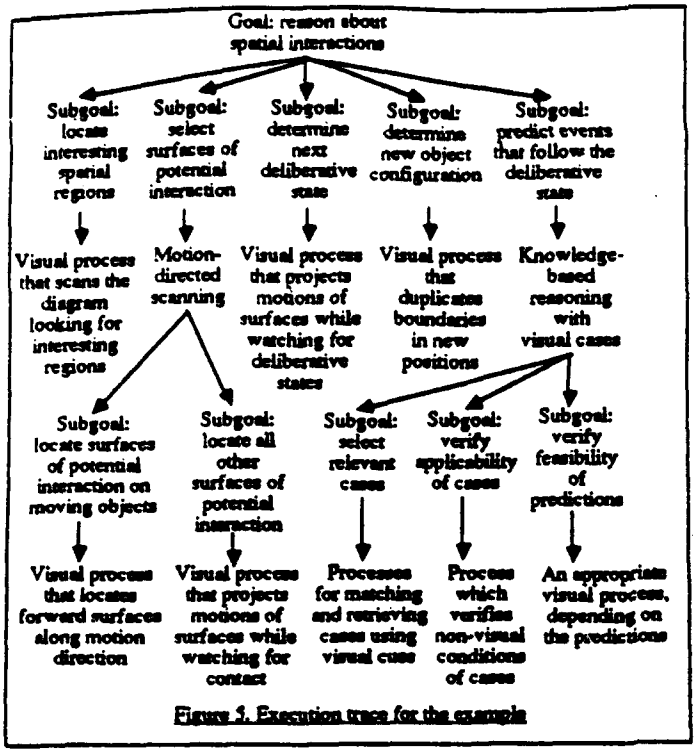


Figure 5. Execution trace for the example

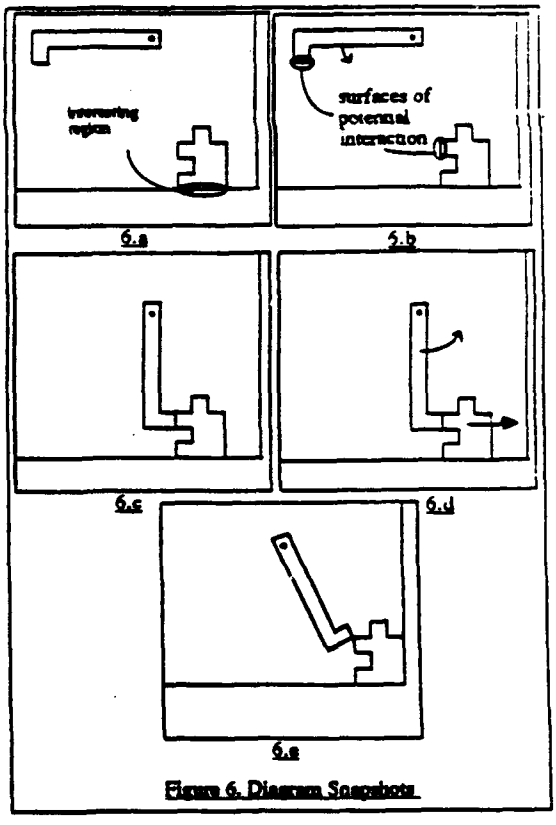


Figure 6. Diagram Snapshots

prediction about the motion of object2 after object1 collides with it, given the problem specification of Figure 2, and then notice how this prediction will change if the specifications were changed to indicate that object1 is non-rigid (say, made of rubber) and that object2 is fixed on surface3. The visual and non-visual parts of a case explicitly capture this aspect. Thus the intent of visual cases is to represent simple chunks of experiential knowledge about spatial events that humans typically have, and to model the blending of conceptual and perceptual information in making spatial inferences. An example of typical knowledge about spatial events is "a rigid object resting on a rigid flat surface, when collided by a moving rigid object, will tend to slide in the same direction". The schematic of a corresponding visual case is shown in Figure 4.

After a deliberative state is detected, visual cases are retrieved and applied to predict events that follow. The retrieval of cases relevant to the spatial configuration in the diagram is based on visual cues. From among the retrieved cases, applicable ones are selected by using information about object properties (which is available as part of problem specification) to verify the non-visual parts of the cases. Events predicted by the applicable cases are further pruned by verifying, through visual processes, their feasibility in the current object configuration. The remaining events serve to guide subsequent steps of reasoning. Since a case brings conceptual knowledge to bear on visual reasoning, this mechanism of inference may be viewed as a computational realization of cognitive penetrability or the influence of tacit knowledge on mental imagery [Pylyshyn, 1981].

2.5 An Example

In this section we present a problem solving episode in some detail. The specification of the problem, which includes a depiction of an initial configuration of objects, an initial motion and relevant non-visual properties of the objects, is shown in Figure 2. The goal is to predict all resulting motions by reasoning about spatial interactions that will occur among the objects. In our computer model control of reasoning is done by a procedure that generates goals and subgoals, and activates relevant processes to achieve them. Thus an execution trace will appear as a tree consisting of goals, subgoals, and processes. The goal generation follows the process model in Figure 3.

Figure 5 shows a partial execution trace for this example. "Reason about spatial interactions" is the

top level goal and it has four subgoals as shown. Consider the first subgoal "locate interesting spatial regions". There is a set of heuristic criteria to locate interesting regions, one of which is that regions representing touching surfaces of multiple objects are interesting. The visual process corresponding to this subgoal focuses on each object in turn, tracks its boundary, and looks for regions that satisfy the criteria. In this example it finds the bottom surface of object2 as shown in Figure 6.a. Next, surfaces that have the potential for interaction are located (Figure 6.b shows the surfaces identified for the current problem) and another process projects the motion of moving surfaces that are identified to have interaction potential while watching for the occurrence of deliberative states. The first deliberative state detected is the configuration in which contact occurs between objects 1 and 2 and the diagram is modified to depict this configuration, as shown in Figure 6.c. The next subgoal is to predict subsequent dynamics of this configuration and this is accomplished by the application of visual cases. Three visual cues (the presence of a rotating object and a stationary object, and the occurrence of a collision between the two) are used to retrieve cases, and visual keys of cases are matched with the current configuration by inspecting its visual representation. The availability of symbolic descriptions as well as diagrams in the visual representation allows the matching of visual keys to proceed at an abstract level without recourse to techniques like template matching. A visual case similar to the one shown in Figure 4 (except that the moving object is undergoing rotation) is found to be relevant and applicable, and the event that it predicts is found to be feasible in the current configuration. Non-visual conditions associated with this case are similar to those in Figure 4 and are easily verified from the problem specification. The resulting prediction is shown in Figure 6.d. As the process model shows, after this step the entire cycle is repeated and in the next detected deliberative state object2 has collided with surface4. This time a case that predicts cessation of motion gets applied and Figure 6.e shows the final configuration.

3 Related Work

In this section we present computational and cognitive research which touches upon imagery, in support of our contention that imaginal reasoning is an emerging research area that is highly promising. Cognitive scientists have demonstrated not only the powerful role of imagery in human problem solving but also the advantages of incorporating similar reasoning capabilities in computer programs. For example, Larkin and Simon [1987] persuasively argue for the computational advantages afforded by diagrammatic representations and perceptual inferences that such representations support, for solving physics and geometry problems. Koedinger and Anderson [1990] describe a model of geometric proof problem solving in which parsing of a diagram of the problem to detect specific diagram configurations is a key step. These configurations then cue relevant schematic knowledge for proceeding with the proof. Visual cases represent a generalization of this idea. Logicians have also noted the power of visual representations. Barwise and Etchemendy [1990] illustrate the role of visual representations in mathematical reasoning through a program called Hyperproof which allows the user to reason using sentential and pictorial forms of information.

Despite intuitively compelling evidence for the use of imagery by humans, there has not been much work in artificial intelligence toward endowing machines with a similar capability. An early program that utilized diagrams was WHISPER [Funt, 1977] which addressed rotation, sliding, and stability of blocks-world structures. More recently, work on using pictorial or "analogical" representations for simulating the behavior of strings and liquids in space has been reported [Gardin and Meltzer, 1989]. Shrager [1990] describes a computational model of understanding laser operation in which reinterpretation processes utilize event depictions in an iconic memory as well as in a propositional memory. The research on computational modelling of the cognitive process of spatial reasoning with diagrams [Narayanan, 1991] is yet another step towards realizing the full potential of imaginal reasoning by computers.

4 Concluding Remarks

We have described a novel approach to reasoning about spatial interactions. Since our aim in this paper has been to provide the reader with an overview of all significant aspects of visual reasoning within the limited space available, the descriptions have been necessarily schematic in character. Further details on components of this approach - structure of visual representations, how visual processes are composed from basic visual operations, indexing and adaptation of visual cases, the computer program that implements this model, etc. - can be obtained from [Narayanan, 1991].

The advantages of using diagrams in this approach arise from the property that spatial information such as obstacles to motion or pathways that guide motion are directly evident in images. Our approach is not only intuitive, but flexible as well.

Objects which have irregular shapes that will make their algebraic representations complex can be represented and reasoned about in the same way as regular objects if diagrams are used.

As Forbus and colleagues rightly point out [Forbus *et al.*, 1987], there can be no purely qualitative method for spatial reasoning. What is required is to integrate qualitative and quantitative methods so that qualitative ones provide approximate solutions that serve to focus the application of quantitative methods to only those aspects of the initial solutions that require more precision or further refinement. With this goal in mind, we are currently investigating the integration of visual reasoning with other qualitative and quantitative methods [Narayanan and Chandrasekaran, 1991].

Acknowledgments

A discussion with Stephen Kosslyn provided inspiration during the early stages and Jeff Shrager and Janet Kolodner helped clarify many issues during the course of this research. We also thank the anonymous referees for their suggestions. This research is supported by DARPA & AFOSR contract # F-49620-89-C-0110 and by AFOSR grant # 890250.

References

[Barwise and Etchemendy, 1990] J. Barwise and J. Etchemendy. Visual information and valid reasoning. In W. Zimmerman, (Ed.), *Visualization in Mathematics*. Mathematical Association of America, Washington D. C., 1990.

[Biederman, 1990] I. Biederman. Higher-level vision. In D. N. Osherson, S. M. Kosslyn, and J. M. Hollerbach, (Eds.), *An Invitation to Cognitive Science, Volume 2: Visual Cognition and Action*, MIT Press, Cambridge, MA, 1990.

[Chandrasekaran and Narayanan, 1990] B. Chandrasekaran and N. H. Narayanan. Integrating imagery and visual representations. In *Proc. 12th Annual Conference of the Cognitive Science Society*, pp. 670-678, Boston, MA, 1990.

[Chapman, 1990] D. Chapman. Intermediate vision: architecture, implementation, and use. Technical Report TR-90-06, Teleos Research, Palo Alto, CA, 1990.

[Finke, 1989] R. A. Finke. *Principles of Mental Imagery*. MIT Press, Cambridge, MA, 1989.

[Forbus *et al.*, 1987] K. D. Forbus, P. Nielsen, and B. Faltings. Qualitative kinematics: a framework. In *Proc. 10th International Joint Conference on Artificial Intelligence*, pp. 430-436, Milano, Italy, 1987.

[Funt, 1977] B. V. Funt. WHISPER: a problem solving system utilizing diagrams and a parallel processing retina. In *Proc. 5th International Joint Conference on Artificial Intelligence*, pp. 459-464, Cambridge, MA, 1977.

[Gardin and Meltzer, 1989] F. Gardin and B. Meltzer. Analogical representations of naive physics. *Artificial Intelligence Journal*, 38:139-159, 1989.

[Koedinger and Anderson, 1990] K. R. Koedinger and J. R. Anderson. Abstract planning and perceptual chunks: elements of expertise in geometry. *Cognitive Science*, 14:511-550, 1990.

[Kolodner and Simpson, 1989] J. L. Kolodner and R. L. Simpson. The MEDIATOR: analysis of an early case-based problem solver. *Cognitive Science*, 13:507-549, 1989.

[Kosslyn, 1981] S. M. Kosslyn. The medium and the message in mental imagery: a theory. *Psychological Review*, 88:46-66, 1981.

[Larkin and Simon, 1987] J. H. Larkin and H. A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11:65-99, 1987.

[Marr and Nishihara, 1976] D. Marr and H. K. Nishihara. Representation and recognition of the spatial organization of three dimensional shapes. AI Memo 377, Artificial Intelligence Laboratory, MIT, Cambridge, MA, 1976.

[Narayanan, 1991] N. H. Narayanan. A study of representations and processes for reasoning visually about spatial interactions. Doctoral Dissertation, Department of Computer & Information Science, The Ohio State University, Columbus, OH, forthcoming.

[Narayanan and Chandrasekaran, 1991] N. H. Narayanan and B. Chandrasekaran. Integration of qualitative and quantitative methods in visual reasoning. In *Proc. 2nd Conference on AI, Simulation and Planning in High Autonomy Systems*, pp. 272-278, Cocoa Beach, FL, 1991.

[Pylyshyn, 1981] Z. W. Pylyshyn. The imagery debate: analogue media versus tacit knowledge. *Psychological Review*, 88:16-45, 1981.

[Schank, 1982] R. Schank. *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge University Press, New York, 1982.

[Shrager, 1990] J. Shrager. Commonsense perception and the psychology of theory formation. In J. Shrager and P. Langely, (Eds.), *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann, San Mateo, CA, 1990.

[Ullman, 1985] S. Ullman. Visual routines. In S. Pinker, (Ed.), *Visual Cognition*. MIT Press, Cambridge, MA, 1985.

Combining Functional Representation and Structure-Based Models for Smarter Simulation

Susan T. Korda, John R. Josephson, Dale Moberg

June 30, 1993

Computer simulation of devices usually consumes significant amount of computational time. Making simulation more efficient and goal-oriented might overcome this problem. The main objective of our model is to provide solutions to this problem by alloying functional representation (FR) with structure-based models for simulation.

Based on the VHDL framework, we selected a very general model of device structure:

- A device is considered to consist of a box with a set of **input** and **output ports**, and possibly a state, and can be characterized by a set of **functions** to convert the values at input ports into specific values at output ports at particular times while possibly changing the **state** of the device.
- A device can consist of a set of **subdevices**, each having its own **functions**, **input** and **output ports**, **internal states**, and also a set of **connections** joining the ports of different subdevices.
- Signals (values) at input ports of a component may trigger signals (values) at output ports, and state change of the component in finite amount of time.
- It is assumed that a connection transfers the value from an output port across a connection to an input port, (or from the input port of a component to the input port of its subcomponent) **instantaneously**.

- Subdevices/subcomponents can be embedded in components of devices to any finite level of depth.

The above described framework applies to a large variety of devices, from software/hardware devices to devices containing pipes and fluids, etc.

In the followings, by simulation, we will always mean simulation based on this kind of **structure/behavioral** model.

Our major goal is to

- show, how FR can be smoothly combined with structure-based models for simulation in a common framework,
- what are the advantages of combining the two, and how FR can control the simulation efforts,
- and what are the tradeoffs between controlling simulation by FR and simulating brute-force by using the structural model alone.

Functional Representation Combined with Structural Model

We supplemented the functional representation language with further elements in order to support structure/behavioral models of *simulation*. We retained most of the earlier additions to the language, like the parametrization of components, and the possibility of defining different relations among them (e.g., *can-run-on(software hardware)*) that we had made in the previous stage of our current project [1].

We added several new features to the representation:

- **Functions** can achieve their result states in response to some starting state, and signals arriving to their input ports. The set of input ports used by the function, and the set of output ports affected by it are subsets of the input and output ports of the component the function is belonging to. The user specifies by means of a **transfer function**, how values at input ports and the initial state of the component transform into values at output ports and a new state of the component in time.

Transfer functions are ordinary LISP functions, they specify **how** FR functions are being achieved.

FR's **TOMAKE** statement that describes the state having been achieved by the function has been retained in the representation for explanatory purposes. The TOMAKE statement provides a **higher level/more abstract** view of the state being achieved by the FR-function than the transfer function does, therefore, it can be more conveniently used to understand how the device is functioning.

An example of the component specification can be seen in figure 1.

- **Functions** can also have their own input and output ports which are some subset of the input and output ports of the device. The user specifies by means of a **transfer function** (written in Common Lisp), how values at input ports and the initial state of the component transform into values at output ports and a new state of the component in time. [1]

  By **transaction** we mean a set of output port values, and a state of the component at some specific time.

  The inputs to the transfer function are the values at the input ports of the function, the actual time, the state of the component, and the projected future transactions of the component (which were created as a result of earlier events associated with the component); and it creates a new set of projected transactions, and a new state of the component at a projected later time. [2]

  Any component may contain subcomponents connected by a set of **connections** (joining ports of different components).

---

[1] **Estimate transfer functions** have also been introduced. They are used for complex components / functions with a deep substructure hierarchy. If the details of the functioning of the given component / function are not important concerning our problem-solving goals, and we do not want to explore it in depth, we still may get an estimate on the functioning of the component/function using an *estimate transfer function*. They work similarly to regular transfer functions, but a short-cut of a possibly deep structure/substructure hierarchy can be provided. The results produced will only be estimate values.

By using estimate transfer functions, simulation can be continued, even if a portion of the functional hierarchy has not been explored.

[2] Signals that are transferred across the device carry an identifier and a timestamp with themselves, so the time they spent in the system can be measured.

```
COMPONENT A/D
PARAMETER
        SF 2000 "Sampling frequency:"
        BS 1024 "Buffer size:"
INPUT
        INIT
        signal
OUTPUT
        sample
FUNCTION
        InitializeA/D TOMAKE STATE active(A/D) = 'running
INPUT
        INIT
TRANSFER-FUNCTION tf-initialize-A/D
        (defun tf-initialize-A/D(inp-list state-var-list parameter-list
func-params time trans-list) ...
)
FUNCTION
        sampling TOMAKE sampled-signal AT-PORT sample
INPUT
        signal
OUTPUT
        sample
BY TRANSFER-FUNCTION tf-sampling
        (defun tf-sampling(inp-list state-var-list parameter-list func-
params time trans-list) ...
)
STATES
        active('idle 'running) = 'idle
        buffer-available = BS
END COMPONENT
```

Figure 1: Component specification for A/D.

- We define **functional groups** as a set of functions of a component. Substructures within a component (set of components joint by connections) belong to one particular functional group. The role of the functional groups is that different subsets of functions may be implemented by different substructures, (but there is no intersection between different functional groups, or the substructures belonging to different functional groups).

  One example for the use of functional groups can be seen in the representation of Main-CPU in our prototype. Main-CPU has four functions:

  INITIALIZE, FFT, Walsh, and filter.

  INITIALIZE has one input port, ON, and one output port, INIT.

  FFT, Walsh, and filter all use the same input port, signal, and each of them have separate output ports. It means that the substructure describing FFT, Walsh, and filter intersects, therefore, these 3 functions are grouped into one functional group.

  INITIALIZE is realized by a completely distinct substructure, without any common part or connection with the three other functions, therefore, it belongs to another functional group.

- As in the usual FR, the **causal processes** can also be specified by which the functions are realized. **OR** relationships are allowed between these causal process descriptions in order to represent alternative ways/causal processes to realize certain functions. (It is similar to the VHDL feature that multiple *architectures* can be specified for one *entity*.) These alternative causal processes may play many important roles, among others they may be useful in order to maintain the *reconfigurability* of the device where alternative causal processes might correspond to alternative configurations (software- hardware assignments).

- Functions or functional groups can separately depend on **conditions** based either on traditional **states** or **relational states** (usually representing some particular software-hardware assignment, but relationship of any kind can be used) under which the function applies. Connections between ports can also be conditioned either individually or in

CONNECTIONS

(4 sample-main OF Main-CPU => sample (FFT OF Main-CPU)

5 sample (FFT OF Main-CPU) => sample (FFT OF fft-main)

6 FFT (FFT OF fft-main) => fft-transform-of-sample (FFT OF Main-CPU))

IF RELATION running-on (fft-main Main-CPU)

Figure 2: A portion of the structural description for the FFT, Walsh, filter functional group of Main-CPU.

any kind of groups. Connections are named so that they can be easily referenced by causal process descriptions.

A portion of the structural description can be seen in figure 2.

- Components can have **instances**. Values of component parameters can be separately set for each instance at its creation, (or any later time). A separate set of input and output ports is created for each instance of the component, but all attributes of the component (functions, transfer functions, states, etc.) are inherited. Connections can be created between ports of either *components* or *instances* (Connections are not inherited by the instances.)

- The state transitions of **causal process descriptions** (*cpd*'s) can be described either by *connections*, or by some *function* of some component, or by some *cpd*'s. The state transition can be conditioned on some assumption (described by a PROVIDED clause), or on a set of configuration-dependent relationships specifying the applicability of the state transition.

  The **state** or the value of some **port** can be made explicit at any point of the cpd. (See example of cpd in figure 3).

Representation of Resource Use

Resources are specified for every component of the device individually. The amount of the resource may depend on some parameters of the component

```
CPD sample-and-transfer
 PORT signal OF REAL-TIME-SPECTRUM-ANALYZER
 CONNECTION 6 OF REAL-TIME-ANALYZER
 PORT signal (samplingOF A/D)
     USING FUNCTION sampling OF A/D
         PROVIDED active(A/D) = 'running
     PORT sample (samplingOF A/D)
 CONNECTION 7 OF REAL-TIME-ANALYZER
     PORT sample (TransferSampleOF bus)
         USING FUNCTION TransferSample OF bus
             PROVIDED active(bus) = 'running
 CONNECTION 8 OF REAL-TIME-ANALYZER
     IF RELATION
     OR
         (running-on (FFT-main Main-CPU)
         running-on (Walsh-main Main-CPU)
         running-on (filter-main Main-CPU))
     PORT sample-main OF Main-CPU
 CONNECTION 9 OF REAL-TIME-ANALYZER
     IF RELATION
     OR
         (running-on (FFT-dsp Main-CPU)
         running-on (Walsh-dsp Main-CPU)
         running-on (filter-dsp Main-CPU))
     PORT sample-dsp OF Main-CPU
END CPD
```

Figure 3: Description of the *sample-and-transfer* causal process.

USING RESOURCE (AT ESTIMATION-LEVEL 1)

(* (* BS t22) 0.84) OF Main-CPU-time WITH TOLERANCE 0.01

USING RESOURCE (AT ESTIMATION-LEVEL 1)

250 OF Main-Memory WITH TOLERANCE 10

Figure 4: Resource-use estimate for function Walsh of Main-memory.

or on the entire component.

We can represent resource use of functions or configurations at different levels of **estimation**. The level of estimation depends on the depth in the representation (estimates at subcomponent level are usually finer than estimates at component level). The accuracy of the estimate is characterized by the value of **tolerance** attached to each resource use specifier: the deeper the level of estimation is, the smaller the values of tolerances should be. (See example in figure 4.)

There are resource use values attached to each software-hardware assignment, these are the rawest estimates. The actual values of resource use are described at the deepest component level (that does not have already subcomponents).

External Conditions

External conditions are represented in a separate section of the combined representation. Specific values at particular ports of the device are allowed to be placed at any given time.

States can be instantiated to specific values in the STATES subsections of components.

Simulation Process

The actual simulation starts with placing the values specified by the external conditions to the appropriate ports of the device.

In every problem-solving cycle, the set of **earliest events** is determined. An *event* can mean either a component that is able to fire, i.e., has all the

needed signals at its input ports, or it can mean a value arriving at some (output) port which can be transferred to another port through a connection.

If there are several events that may happen at the same time, the causal process descriptions can be used to order those possible events (as will be discussed later in detail).

If a particular component is due to fire, (possesses an active event) and all of its conditions are satisfied, then either its transfer function will be invoked (if the component does not have subcomponents), or the substructure of the component will be traversed, and the input signals will be transferred through the connections. The simulation process is complicated by goal-dependent decisions [3] as it will be discussed later.

The simulation terminates if

- either the predefined simulation time has been elapsed,
- or there are no more transactions left in the system, i.e. no more event can happen,
- or our problem-solving goal has been satisfied.

Advantages of the Simulation - Functional Representation Symbiosis

Placing FR at the top of the simulation model results in representational benefits, on one hand, and also provides significant advantages in controlling the problem-solving process, on the other hand.

Representational Benefits

The FR language makes a very important distinction possible between *functions* and *components* (as parts of the *structure* of the device).

The simulation language, VHDL, can also represent functions, the *entities* in VHDL can represent either functional or structural elements. Nevertheless, VHDL cannot make any *distinction* between the two, cannot describe how

[3] e.g., estimating resource use and checking on the estimates in each step if the goal is deciding resource-satisfiability

structural and functional elements are related to each other, and it may be a major drawback for several purposes (for example, diagnosis). [4]

Another advantage of using FR is that *causal processes* that describe the behavior of the device are represented explicitly. Though, this information is included also in the VHDL description of the device, in the form of how signals traverse the system, it is hidden within the representation, and cannot be used efficiently for controlling problem-solving. In particular, we would like to use causal process descriptions to control simulation based on pre-defined **problem-solving goals.**

Another advantage of our representation is that causal process descriptions, or even the structure of the device (the connections between the parts) can all be conditioned either on assumptions, or on configuration-dependent relations, making explicit representation of *reconfigurability* possible.

Using the Combined Model for Problem-Solving

Our main objective is to reduce simulation efforts based on the goals we would like to achieve by the simulation. Currently, our first goal was to find out whether a given configuration would work without exhausting any of the resources, the second goal was to determine the values of the signal at a pre-defined set of ports.

Deciding Resource-use Satisfiability

Our strategy was the following:

- In the first step, we determined the set of **critical resources** based on the highest-level resource-use estimates. [5]
- Simulation has been applied in an intelligent fashion to find out more about critical resources (it will be discussed later in detail).

[4] **Example:** Exactly the same structural port may be used by more than one function of the component. In the ideal case, the different kinds of uses of that port can be examined independently. However, if the component fails, the two uses of that port can conflict with each other.

In our prototype, the *signal* port of Main-CPU is used by 3 different functions, FFT, Walsh, and filter, every other port is used by exactly one function.

[5] **Critical resources** are those for which we cannot decide whether they can or cannot be satisfied based on earlier information.

- If one of the critical resources has been exhausted, i.e., we know for sure that the given configuration cannot function, the simulation will be stopped immediately.

- If one of the critical resources turns out to be sufficient without any ambiguity, it will be removed from the set of critical resources.

- If there are no more critical resources left, the simulation effort will be stopped. [6]

- If the simulation terminates due to either of the termination conditions described earlier in the 'Simulation Process' section, then the remaining set of critical resources cannot be resolved by the simulation, only the estimated value and the tolerance can be decided.

Determining Values of Signals at a Predefined Set of Ports

This goal is simpler to implement than the previous task. Simulation should be performed that terminates if every goal has been satisfied, i.e. every interesting port value has been determined.

In case of both goals, our effort has been focused on **intelligently controlling simulation** based on FR. We developed two different strategies for this purpose, and applied those in a combined fashion:

- One method was to control the **depth of reasoning** based on problem-solving needs (i.e., not to descend deeper into the component-hierarchy than required).

- The other strategy was to **avoid** following **irrelevant simulation paths** which do not contribute to our problem-solving goals.

Controlling the Level of Reasoning

Most devices can be examined at many different levels of detail, from the level of the entire device to the level of components, subcomponents to the smallest represented parts. Significant amount of computational time can be saved if it

[6] It may depend on our additional simulation goals whether outputs should be created as a result of the functioning of the device.

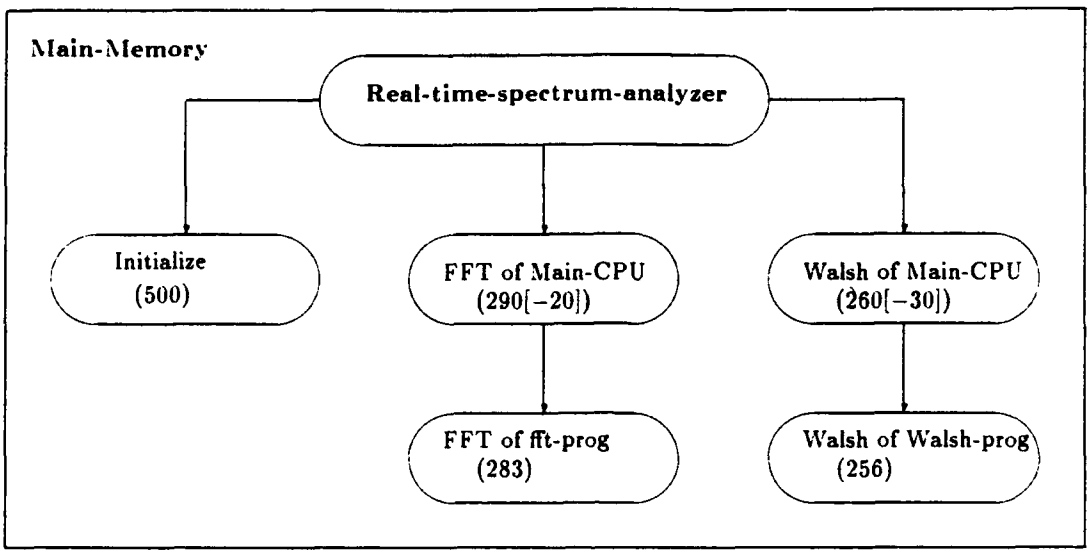


Figure 5: Resource use hierarchy for the resource Main-Memory. The topbox means the entire device, each of the rest of the boxes mean a function of certain component, and its resource utilization, or resource utilization estimate (in parentheses) with the tolerance (in brackets).

can be decided during problem-solving where it is worth skipping the details. Our strategy is that we should go deeper into the component/functional hierarchy if and only if we can gain valuable information by that concerning our particular simulation goal (i.e., related to at least one of the critical resources, or interesting ports).

In case of resource-use satisfiability, the implementation of this strategy consists of two main steps:

- In one step, a set of **resource use hierarchies** has been built for each resource before beginning the simulation process. This set contains every function that uses the given resource, parent/child functions are mutually linked to each other.

  An example for a resource use hierarchy can be seen in figure 4.

- Every time the simulator discovers a new substructure which could be explored, it will decide whether it is worth exploring it by analyzing

the resource use hierarchies of the critical resources. If it has been decided that a short-cut can be taken, i.e., going into more details would not provide any additional information on any of the interesting resources, then the *estimate transfer function* of the current function will be used (that the simulation can continue).

In case of determining port-values, a short-cut can be taken while traversing a module if no interesting port resides within that module.

Controlling Simulation By Means of Causal Process Descriptions

In the first run, we applied simulation without the explicit use of cpd's. We discovered several inefficiencies:

- Let's assume that the device contains two causal processes, A, and B, and B causally depends on A. Assume that each of them can be executed by the simulator simultaneously, i.e., there are potential transactions with the same timestamps in the waiting queue belonging to each of the paths. The simulator selects one of the transactions randomly to execute, therefore, let's assume that the transaction on path B will be selected, and the same happens also in the subsequent steps. Then, at one point, the transaction picked on path B will fail because of the causal dependency, and the simulator will continue on path A. [7]

  We could have saved this failing step by **ordering** transactions according to the causal paths they are belonging to. [8]

- Another source of inefficiency is if simulation is run on causal paths that are completely **irrelevant** of the problem-solving goal, i.e., if some processes are followed that

[7] We had an example of this situation in our prototype: Our prototype consists of two major causal processes, the first one initializes everything in the device, and the second one performs the actual functions of the device (sampling, then calculating FFT, Walsh- transforms or filter the signal, etc.). If we attempt to do sampling before A/D (that performs sampling) has been initialized, this step will fail. We may waste valuable computational steps by attempting it more times.

[8] An ordinary simulator checks on the conditions of all the earliest events in each cycle, and picks the one with satisfying conditions. Our simulator, on the other hand, intelligently knows which event to pick without this condition-checking.

- do not use the critical resources, and
- no other processes of the device depend on them which use any of the critical resources. [9]

In order to overcome the above described inefficiencies, we can use FR's **causal process descriptions to control simulation.**

The implementation of this strategy requires pre-processing of the representation before the problem-solving process can start: cpd's are pairwise analyzed, whether there is

- any **causal dependency** between them
  - either some assumption required by one process is established by the other one,
  - or some port-value required by one of them has been filled by the other one.
- any **resource dependency** between them (i.e., do they use the same kind of resource).

The kind of preliminary analysis of FR's causal process descriptions as described above is **independent of our particular problem-solving goals.**

Simulation can be controlled by the cpd's in the following way:

- As discussed earlier, in every simulation step, the set of earliest transactions is selected (that can happen simultaneously). For every transaction, we determine the particular causal process it belongs to, then
- each of the cpd's will be checked whether it is **relevant** to the problem-solving task, if not, then, it will be removed from the set of active events,

[9] **An example from our prototype:**
It turns out at the rawest estimation-level that MAIN-MEMORY is our critical resource. In our configuration, *filter* runs on DSP. From the causal dependency analysis of our device, it will be clearly seen that the *filter process* does not influence our decision on MAIN-MEMORY-use: it does not use MAIN-MEMORY itself, and no other processes depend on it that use that resource.

- the cpd's of active transactions are checked pairwise for **causal dependencies** between them, and they will be ordered to be acted upon accordingly.

One important feature of the above described implementation strategy is that **goal-dependent** and **goal-independent** steps are well-separated:

- The parsing and preliminary analysis of FR (analysis of cpd's) are goal-independent, the simulation process itself similarly.
- During the simulation process, some **goal-dependent decision-functions** should be called in order to control simulation, to
  - order simulation steps, and
  - decide possibilities to prune irrelevant simulation paths.

Tradeoffs between Controlling Simulation by FR and Pure Simulation

As we could see in the previous section, controlling simulation by FR costs plenty of overhead. However, a significant part of additional computation can be precalculated, and most of the simulation-time overhead can be economized by additional prior computations.

Whether it is worth the extra effort, depends strongly on our **goals**, and the kind of device being simulated. The more paths can be weeded out before performing simulation on it, the more economically our suggested method can be used.

If we have a few pointed goals, then it is almost sure that using FR will lead to significant savings. On the other hand, if a vast amount of information should be collected, we may need to think about some other, more appropriate way to reduce simulation efforts.

References

[1] Susan Toth Korda, John R. Josephson, Dale Moberg: Representing Function for Reasoning about Software-Hardware Reconfiguration, LAIR tech. report, 1992.

Representing Function for Reasoning about Software-Hardware Reconfiguration

Susan Toth Korda, John R. Josephson, Dale Moberg

June 30, 1993

A prototype system for automated generation and testing of software/hardware configurations can benefit from using device representations that focus on functions. These capabilities can support designers in analyzing the designed device. For example, our benefit is that some major characteristics, like resource uses or fault-tolerance can be examined without performing detailed simulation. Based on our ability to generate and test configurations, we gave an example of how automated reconfiguration can take place which can support fast reconfiguration in case of component-failures. Another interesting benefit of our problem-solver is that functions can be prioritized so that if not all functions can be achieved, a degraded mode of operation may still be possible.

1 Using the Functional Representation Language for Automated Reconfiguration

Understanding a device means, in part, understanding how its functions are achieved by way of the functions and behaviors of its components and their subcomponents. This kind of knowledge can be used, for example, for diagnostic reasoning [1], causal explanation of diagnostic conclusions [2], control of simulation [5], or debugging of computer programs [3]. In case based design, the known functions of pre-stored designs can provide indexes to assist with finding the closest pre-stored design case to the desired design [4].

Systems comprised of software running on hardware can be considered as devices. The hardware elements and the programs running on them are the components of the device, and also the information that moves around and is transformed during processing is special kind of component (we call such things *'movable substances'*). Our work is based on extensions to the Functional Representation language (FR), first described by Sembugamoorthy and Chandrasekaran in [1]. The FR language describes a device as something which has one or more known functions. The device is made up of components and the main device achieves its functionality by employing the services of its components.

FR was originally devised for physical devices, but as [3] has shown, programs can also be considered as devices. In our domain, a program can usually run on multiple hardware components, i.e. the same functionality can be achieved in several ways.

By a particular **configuration** of the device we mean a specific assignment of software to hardware.

Reconfiguration means reassignment of software to hardware which may be needed as a consequence of a hardware failure, or in order to achieve a better utilization of resources.

Estimating the **resource use** of a particular configuration, and reconfiguring the device around faults may be major concerns of users. We can represent the device either independently of the particular configuration, i.e., regardless of which software is actually running on which hardware element, or the representation may reflect the specific association between the software and the hardware the program is running on.

We supplemented the Functional Representation language with further primitives which make it more suitable for reasoning about devices consisting of software and hardware components and solve the desired problem-solving tasks. A small **prototype** has been created (see figure 1), and the enhanced representation has been used to solve several problems:

- Check the consistency of the configured device:
  - Will the desired functions be achieved by the device ?
  - How much of the resources are used on the average during its functioning ?

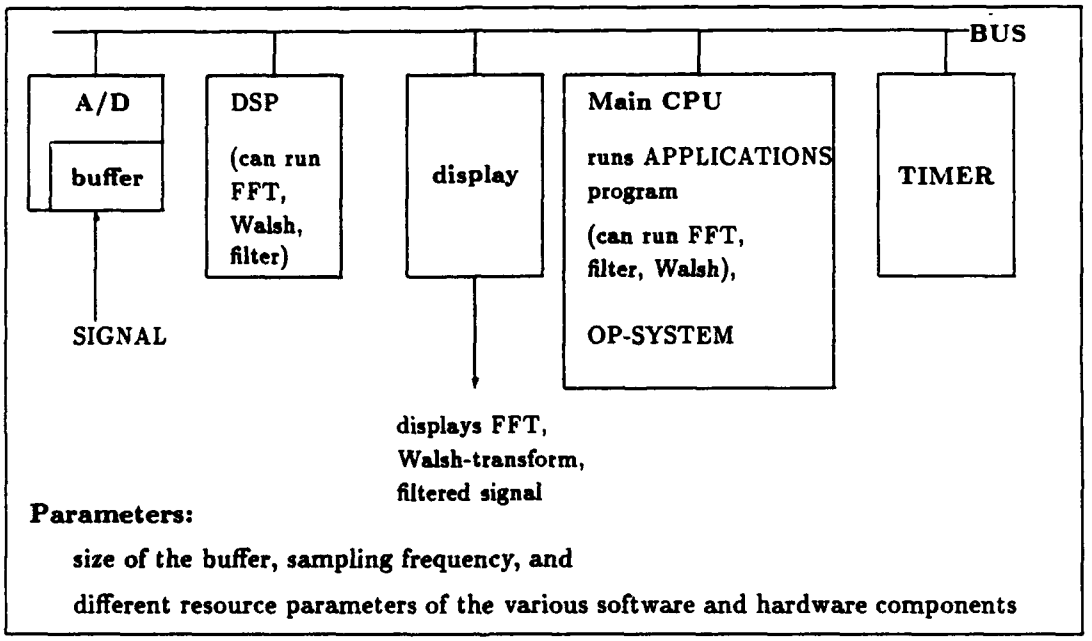


Figure 1: Simple signal-processing device consisting of an A/D converter, a digital signal processor, a display, a main computer, and a timer represented in the prototype.

- Check all possible alternative configurations, and figure out which of the desired functions are achieved by each of them, and estimate the resource use for each.

- Try to find all possible configurations if one or more of the components failed. Assign a priority order to functions and try to achieve high priority functions first. Determine which functions can be achieved in this degraded mode (if not all of them are achievable).

2 Extensions to the Functional Representation Language

The Functional Representation language represents the **structure** of the device in terms of components and relations among them; the **functions** of the device and its subdevices at several levels of abstraction, and provides a **causal process description** (in other words, **behaviors** of the device) of how the various functions are achieved by the components/ subcomponents of the device while assumptions made and generic knowledge used are also represented explicitly. The device and its components can be in different **states** during their operation, and a particular behavior of the device can be described in terms of a chain of state transitions.

Our particular field of application, (software/ hardware environment) required several extensions to be made to the existing Functional Representation language:

- **Parameters.**

  The components of the device may have parameters, the values of which are determined externally by the user (size of the buffer, sampling frequency, etc.). A variable-name, a default-value, and some description (text) is assigned to each parameter.

- **Resources** are assigned also to components, the **type**, **amount** and **unit** is specified for every resource (example: 128 blocks of RAM). The amount of the resource may be expressed as a function of the parameters of the components.

- **Resource uses** are attached to behaviors (the amount of certain kind of resource used during some state transition is specified along with the particular state transition).

- **OR relationship** is used between alternative behaviors which achieve the same state transition. The alternative branch being followed depends usually on a configuration-related condition (how the software is assigned to hardware). Each alternative behavior may require a different set of software/ hardware assignments.

- Two specific **relations** for the software/hardware context, **running-on** and **can-run-on** are attributed special meaning:

  - **Running-on** means that the given software can run on the set of hardware components, and it is initialized, loaded, started, etc.

  - **Can-run-on** means only the *ability* of the software to run on the particular hardware.

The meaning of these two relationships are hard-coded in our current representation language. It would be nicer in the future to extend our representation with new primitives which make it possible for the user to define his own types of relationships.

3 Discussion

In the followings, we will discuss, how the reconfiguration task (finding alternative configurations) is supported by representing the functions of the device, in other words, how strongly FR is used in our problem-solving tasks.

The first task checks the consistency of the function/behavioral description of the device and estimates the average resource use by following causal paths in the representation. In this task, the FR is used only as an organization principle to describe the device in terms of functions/ components/ subfunctions, etc.

Estimated resource uses are attached to functions, behaviors, achieving those functions, or the state-transitions, constituting behaviors, at different levels of the functional representation. In order to estimate the overall

amount of resources needed to achieve some function, the FR is used to determine all the lower level functions, behaviors associated with that function, so that the resource uses belonging to those lower level functions, behaviors, state-transitions can be accumulated.

Checking the consistency of the FR of the device means crossing the FR hierarchy associated with each function of the device, and following all the possible paths, how it is realized. It should be checked whether each *behavior* achieving some function is defined in the FR, and really achieves the desired function, and all the *states* constituting the corresponding state-transitions are declared for the appropriate components.

The second task **generates** all possible **alternative configurations** combinatorially, and tries each of them individually. The method of creating alternative configurations builds fundamentally on the FR. Two configurations are considered as alternatives based on their **functionality**.

Alternative configurations are generated combinatorially, in all possible ways. The number may be very large, the system does not have any knowledge to guide generation or to select the best alternatives. Therefore, the system must be supplemented with further knowledge for this purpose.

The implementation first creates a list of possible assignments *(function, software-component, hardware-components)* triples, based on the *'can-run-on'* and *'running-on'* relations specified in the RELATIONS section of the representation.

The set of possible assignments is *filtered* based on the desirable functions, using the functional representation: only those possible assignments remain in the list which contribute to at least one of the functions to be achieved. It is implemented by crossing the FR hierarchy associated with each desired function.

Alternative behaviors (which achieve the same state transition in different ways) are specified by *OR relations* between behaviors realizing functions or state-transitions (as mentioned in the previous section as an extension to the FR language). Alternative behaviors are used to create alternative subsets of possible assignments (of software to hardware components), each subset belonging to one of the behaviors (see figure 2). Assignments belonging to a behavior are parts of the *conditional description* of the behavior, (i.e. provide conditions for the validity of the behavior).

$$\left(\left(\begin{array}{c} \text{A - running-on - B} \\ \text{C - running-on - D} \end{array} \right) \textbf{or} \left(\begin{array}{c} \text{A - running-on - X} \\ \text{Y - running-on - W} \\ \text{Z - running-on - T} \end{array} \right) \textbf{or} \left(\begin{array}{c} \text{A - running-on - B} \\ \text{M - running-on - N} \end{array} \right) \right)$$

and

$$\left(\left(\begin{array}{c} \text{D - running-on - E} \\ \text{F - running-on - G} \end{array} \right) \textbf{or} \left(\text{H - running-on - J} \right) \right)$$

Figure 2. Alternative sets of possible assignments.

Actual configurations are selected in all possible combinations using a brute-force approach based on the alternative subsets of possible assignments, one assignment is picked from each set of alternatives, in all possible ways.

The third problem is to **design around faults**. It is similar to the previous task, but it generates only those configurations which do not utilize the faulty component(s). Before creating alternative subsets of possible assignments, all assignments are checked for faulty components, and assignments containing one or more faulty component(s) are removed from the list.

Desirable functions may be organized in a **priority order**. The possible assignments are ordered based on the priority order of functions: The functional representation of the device is used to decide which possible assignments are associated with each of the functions, and possible assignments related to the highest priority function will be at the beginning of the possible assignment list, etc. The goal of this ordering is that we want to make sure that we cannot run out of some resource required for the achievement of some high priority function in consequence of the resource use of some lower priority function, i.e. we want to ensure that higher priority functions are achieved before lower priority functions.

4 Conclusions

We have shown, how we can reason about software/ hardware configuration, and reconfiguration based on the Functional Representation of the device.

Our program calculated the estimated average resource use of the device during its functioning. Our representation contains only average values, since the device is described at a very high level, the function level, the processes underlying the functions are not represented. Therefore, a detailed quantitative simulation cannot be completely replaced by using the functional representation, but FR may help focus simulation on the most interesting, and critical possibilities.

References

[1] Sembugamoorthy, V., and Chandrasekaran, B.: Functional representation of devices and compilation of diagnostic problem solving systems. 1986.

[2] Keuneke, Anne M.: Machine Understanding of Devices. Causal Explanation of Diagnostic Conclusions. PhD thesis 1988.

[3] Allemang, Dean T.: Understanding Programs as Devices. Ph.D. thesis, 1990.

[4] Goel, K. Ashok: Integration of Case-Based Reasoning and Model-Based Reasoning for Adaptive Design Problem Solving, PhD thesis, 1989.

[5] Sticklen, J.: MDX2, An Integrated Medical Diagnostic System. PhD thesis, The Ohio State University, 1987.

Multilevel causal-process modeling: bridging the plan, execution, and device-implementation gaps

Keith Levi

Maharishi International University
Computer Science Department
Fairfield, IA 52557

Dale Moberg

The Ohio State University
Laboratory for Artificial Intelligence Research
Columbus, OH 43210

Christopher Miller and Fred Rose

Honeywell, Inc.
Systems and Research Center
Minneapolis, MN 55418

1. INTRODUCTION

Knowledge-based systems typically require extensive knowledge about the domain in which they are to operate, including knowledge about the objects and systems which exist within the domain, their properties, architectures, capabilities, and interactions. This is particularly the case for systems that require complex reasoning–e.g., systems that perform explanation, model-based and qualitative reasoning, multilevel and integrated reasoning, design/redesign, training and tutoring, and certain systems for knowledge acquisition and machine learning. Acquiring and representing knowledge at the breadth and depth required for these approaches is especially challenging in complex, real-world domains such as industrial and aerospace applications. On the other hand, it is commonplace in these domains for equipment to be well documented, both before and after manufacture, with detailed specifications about its functionality and performance characteristics. It will soon be the case that most of these equipment specifications will be written and developed as executable software simulations in hardware description languages such as the VHSIC Hardware Description Language (VHDL).

VHDL was developed during the early 1980's as a standard, vendor independent description of hardware. With the growing complexity of microelectronics, it was clear that schematic representations would no longer be adequate for description and subsequent support. VHDL was standardized by the IEEE in 1987, thereby giving the industry a standard hardware description language. VHDL supports multiple levels of abstraction, but is most useful at describing the functional or behavioral aspect of hardware. It allows the designer to write a high level description of a piece of hardware. This description can then be used for simulation, analysis, and by a multitude of other tools that need to understand the structure and behavior of a specific piece of hardware.

Clearly, it would be of great benefit for knowledge-based systems to obtain some of their prerequisite knowledge from such sources. As one prominent researcher in the area of explanation based learning speculated: "We foresee a future in which manufacturers of component equipment themselves provide descriptions, in some standard knowledge-based formalism, of the functionality of their product just as today they provide technical descriptions in a form understandable to [equipment] designers. As the manufacturer makes available refined versions of installed equipment, the old knowledge-based description of the component is simply supplanted with the new. The knowledge engineer need only oversee incorporation of the new knowledge insuring that there are no negative interactions that harm overall system performance." [1]

Realising this vision of the future involves several technical challenges. One challenge is it to bridge the representational gap between the syntax of hardware design languages and AI systems. We do not see this as a significant technical hurdle because several studies have already demonstrated that this can be done. These investigations have

used AI reasoning systems to reason about hardware models [2,3,4,5] or have done model-based reasoning using VHDL [6,7].

A more significant challenge involves bridging a gap in semantics. By this we mean going from reasoning at a relatively well-defined and self-contained level of hardware devices to planning and executing actions in the real world involving uncertainties associated with external agents and incomplete, uncertain, and incorrect information. All the previous studies we have cited principally reasoned about behaviors and functions of a device and its subcomponents, not about the function of the device and its subcomponents in the real world. Although one can envision modeling the whole world as a device in which any given device is a subcomponent, this has generally not been a tractable approach. It is this gap between modeling plans and their execution in the real world and modeling the internal behavior of devices in hardware design languages that is the focus of this paper.

1.1 Domain knowledge requirements for real world applications of EBL

Although there are many AI reasoning systems that could benefit from having a detailed source of domain knowledge, our motivation for pursuing this research has come from an application of machine learning in the domain of pilot aiding. DARPA and the U.S. Air Force's Pilot's Associate (PA) is a coordinated suite of five expert systems which cooperate to aid the pilot of an advanced tactical fighter. Intelligent associate systems such as PA [8] support real-time planning and decision making in dynamic and evolving situations. An automated associate aids a human pilot in the performance of a mission by obtaining and compiling information, recommending courses of action and, in some cases, actively performing certain mission tasks.

PA's problem-solving architecture employs distributed, cooperating modules. Situation Assessment and System Status subsystems combine various functions involving monitoring the internal aircraft and external ground and air environments. Mission and Tactics Planners coordinate the planning, scheduling and routing functions. A Pilot Vehicle Interface subsystem serves to assist in execution functions by delivering information to a human pilot, making adjustments to information supplied in accordance with his apparent intentions, and monitoring actions accomplished.

We have been conducting a research program [9,10,11,12,13], titled Learning Systems for Pilot Aiding (LSPA), to automate portions of the off-line process of incorporating new information into the knowledge bases of PA and then propagating pertinent changes to various PA modules. An initial study [11,12] recommended Explanation-Based Learning (EBL) as having the best payoff for knowledge acquisition in this domain. EBL requires a *domain theory* of facts and relationships in the target domain, as well as a *learning instance*, a new case not previously understood by the system [14]. In LSPA, the learning instance can come from records of actual pilot behavior in a flight simulator. The domain theory consists of a set of relatively-unchanging facts about the physical capabilities of the aircraft and the world which it inhabits. When these facts and relationships change, such as when new technologies are incorporated, the domain theory must be updated. Given an up-to-date domain theory, however, novel tactics can be learned automatically from a record of the flight in which they occurred. The system builds an explanation of how a known goal was accomplished via its understanding of the world. This explanation can then be generalised to produce a new plan to be incorporated into PA.

A critical issue for any application of explanation-based learning is the scope and nature of the domain theory required for generating explanations. Our domain theory rules presently cover much of a relatively self-contained portion of the PA domain, the domain for defeating surface-to-air missiles. This is, however, only a small portion of the domain that the PA system deals with. While our domain theory is successful as a feasibility demonstration, the issue of whether we can scale-up our system to cover the entire PA domain remains an important open question.

If the required domain theory for EBL is sufficiently large then it might be more work to build this domain theory than it would be to simply build all of the tactical plans directly. We believe, however, that this will not be the case. An EBL domain theory is essentially a model of the primitive functionality of the aircraft and its environment. Tactical plans, in contrast, model all known behaviors arising from the functionality of the aircraft and its environment. This is analogous to a set of axioms and the set of all possible theorems that can be derived from the axioms. The former

is typically a finite set and the latter is typically infinite.

Thus, the strategy of the LSPA system is that if we can create a domain theory (i.e., set of axioms) we can then automate important parts of the knowledge engineering process for creating a PA. The issue that we are now addressing is whether we can successfully scale-up our domain theory about the pilot-aiding domain.

A full domain theory should be integrated across the different PA expert systems. This will require different perspectives of common events, equipment, and actions for the different modules [9]. Another important requirement, which is the main focus of this paper, is that this integrated domain theory should also be connected to the aircraft systems. The domain theory will necessarily be heavily based on assumptions about the aircraft and its subsystems. Aircraft are frequently retrofitted with new equipment and capabilities. Explicitly modeling functionality dependencies of the aircraft will be necessary for the domain theory to be easily updated as the aircraft evolves.

1.2 Outline of paper

In the remainder of this paper we will report on preliminary work investigating whether an AI high-level mission performance model can usefully connect to and share information with a low-level VHDL model of hardware components. A novel aspect of this work is the degree of separation of the mission performance functions from the isolated behavior of the hardware components. Other researchers have investigated diagnostic reasoning using VHDL models. There is a significant gap, however, between the level of reasoning that is done in such diagnostic settings and our mission performance setting. The functionality of the diagnostic context is typically closely connected to the internal behavior of the given device. In our domain we are reasoning about functionality of hardware within a much broader context. Any given piece of hardware might be involved in many different mission functions. Representing these connections between the mission functions and the hardware functions is a significant challenge in itself because there are so many layers and interconnections.

We expect that representing these connections should have significant benefits in two directions. First, there should be *top-down* benefits that go from the mission performance model to the device model. An appropriate representation should enable us to develop reasoning procedures that can derive hardware specifications from the mission performance model. Similar reasoning procedures could verify that a given hardware design can actually fulfill mission performance requirements, or at least systematically generate VHDL simulation testbenches based on the high-level mission performance model. A second direction for benefits will be *bottom-up* from VHDL models of hardware components to the mission performance model. These benefits will include instantiating constraints on the performance model from VHDL models of the hardware.

In the rest of this paper we will present preliminary work in these three areas. We will first present an example of a representation that is well-suited to capturing the multiple layers and interconnections involved in relating a high-level mission performance model to low-level VHDL device models. We will then present examples of top-down and bottom-up connections between our high-level mission performance model and a VHDL model of a graphics processor for cockpit displays.

2. FUNCTIONAL REPRESENTATION AS AN INTEGRATIVE GLUE

Much of our thinking about ordinary devices and how they work involves viewing those objects as contributing to various causal processes that make the device realize its functions. Causation has been called "the cement of the universe," and few would contest the centrality of causal relations, and the role they play in decomposing systems into subsystems, components, subfunctions and constituent causal processes.

One organization of our causal knowledge of a system, especially a complex system such as a pilot flying an aircraft, is from overall functionality down to the subsystems with their subcomponents and their contributing functionalities. A functional decomposition of aircraft flights is primarily of value to show how to move from higher levels of functional description to lower levels of causal processes that realize functions of interest. A decomposition in which the functional parts are both represented and related then provides the "integrative glue" for subsequent problem

Device: Ownship

Function: Safe-from-Ground-Based-Missile(PopupSamSite)

If: (Noticed PopupSamSite Obstacle to CurrentRoute)

Prevent: (SamSite launch at ownship)

Until: (Past PopupSamSite)

Provided: (And
(No authorization to destroy encountered PopupSamSites)
(No directive to destroy encountered PopupSamSites))

By: (XOr
Prevent-Track-PopupSamSite
Break-Track-PopupSamSite
Prevent-Fire-Control-Solution)

Decomposition: If radar in search mode do chaff, electronic warfare and/or cloaking.
If in track break track.
If in fire-control break fire-control and (concurrently) break-track

Figure 1: The Safe-from-Ground-Based-Missile(PopupSamSite) of Ownship

solving across the levels. If we want to know how an equipment change, for example, might effect parameters governing a maneuver for accomplishing some flight goal, the dependencies marked in a functional decomposition can show us what to look for, and how to figure out what has changed.

Functional Representation[20,2] was developed as a representational scheme for the causal processes that culminate in the achievement of device functions. Functional Representation (FR) explains how a device function is achieved. The function of the overall device is described first and the behavior of each component is described in terms of how it contributes to the function. FR has been used for diagnosis, design, redesign, explanation and prediction[2,15,16,17,18,19,20,21,22]. Previous investigations have also considered some aspects of how FR can assist in machine learning, knowledge acquisition and knowledge compilation[23,24,25].

Functions and causal processes are closely related to goals and plans for achieving the goals. This suggests that FR should be a natural represention for pilot-aiding domain theory. Since FR was originally developed for representing hardware devices, it also should be a natural representation for connecting to the device level of an aircraft, i.e., in this case VHDL models. The remainder of this section presents a preliminary set of functions and causal process descriptions (CPDs) that model one of the scenarios from the LSPA program. The purpose of this model is to give a sense of the representation that will be required.

The goal of the scenario we will consider is to be safe from a surface-to-air missile threat. This goal, or desired result, is a high-level function (of a tacit aircraft-mission device). We will model the plans associated with this function as causal processes to achieve or maintain this result.

In the pilot-aiding domain, it is quite common for several plans to be available for a given goal–even a quite specifically delineated goal. This feature means that to model the goal-to-plan relation functionally, the usual FR formalism needs to be augmented. The addition of Or, Xor, Concurrently operators is one addition required for devices having a situation dependent reconfigurability (most obviously in those devices that are at least partially controlled by agents). When such constructs are used, it is useful to have specific Decomposition Rules to indicate what CPD should be used when. We will illustrate this augmentation to FR with the function described in Figure 1.

Figure 1 provides a function frame for a specific goal of being safe from ground-to-air missile threats. A "Popup-SamSite" is a previously unknown ground-based missile site. No mission planning or routing information would have indicated its existence, and its presence is then an unanticipated hasard of some mission. We are presuming that the

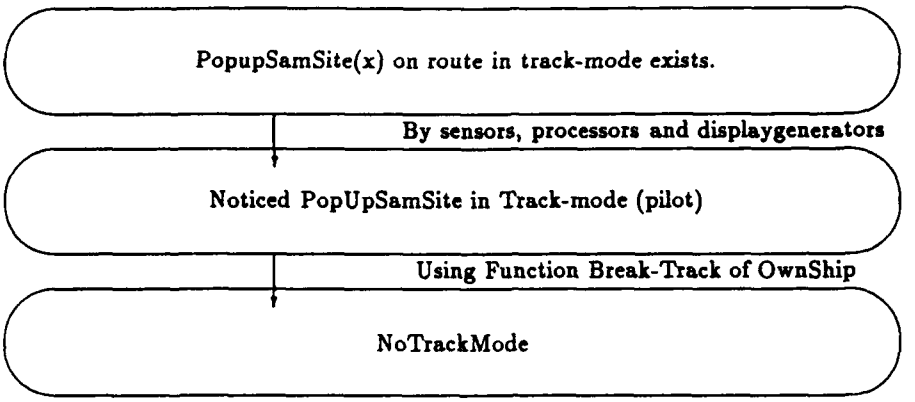


Figure 2: CPD: Break-Track-PopupSamSite

semantics of Safety goals are linked to mainly defensive measures; the functionality of being a defensive safety goal is indicated in the Provided field that suggests that no preemptive destructive goals are appropriate or warranted. In general, the Provided fields are used to indicate the *Standard Operating Conditions* under which the current CPDs are assumed to work. The semantics of these fields are especially important for deciding that a malfunction exists. This is because the truth of Provided fields are presuppositions for raising the question of whether and how a malfunction has occurred.

The resultant state that "defines" the Safe-from-Ground-Based-Missile(PopupSamSite) of Ownship Function is that a launch of the PopupSamSite is prevented. Preventing a launch is essentially a matter of denying the Sam-Site a means of obtaining a state that enables a launch. In this illustration, the focus is on those causal processes that block a launch-capable state by denying appropriate radar information.

There is no unique causal process that invariably is appropriate for preventing the PopupSamSite launch. The prevention functionality is an abstraction over a potential range of causal processes, any of which can prevent launch. Unlike simple devices where a given function depends on some fixed causal process for its realisation, a device involving an agent operating in an uncertain environment typically needs to be represented as having a capability to exercise a range of optional processes. In the current illustration, for example, different actions are appropriate depending on the state of the radar of the PopupSamSite. We will imagine that radars are, if working, in one of three mutually exclusive modes with respect to us: search, track, or fire-control-solution mode. The primary differences concern the degree to which information has permitted the radar installation to notice and predict ownship's (our aircraft's) path. Thus different actions will be appropriate for different modes. For example, if the radar is in search mode, we desire it to remain so and only engage in measures that will impede the radar obtaining a track.

In this illustration, we will not pursue a breadth first expansion of all possible CPD expansions, but will instead imagine a traversal of our model that proceeds to greater and greater levels of detail. We next consider the CPD of Break-Track-PopupSamSite, which provides one way to prevent the PopupSamSite launch, and thus achieve the safety goal.

In Figure 2, we present the next layer of decomposition in a CPD for Break-Track-PopupSamSite. The initial causal state in this process is that there is a PopupSamSite that is in track mode. Then by a (quite complex) causal process that we will not expand here, sensors detect signals that are processed and made available to display generators that illuminate surfaces for the agent to sample. This sampling process by a pilot normally results in the pilot noticing

Device: Ownship
Function: Break-Track
If: (PopupSamSite(LocationIdentifier) in track mode)
Makes: (PopupSamSite(LocationIdentifier) in search mode)
By: (Or Constant-Distance-Behavior ElectronicMeasures)
Decomposition: If Type(PopupSamSite 1) then Constant-Distance-Behavior.

Figure 3: Break Track Function

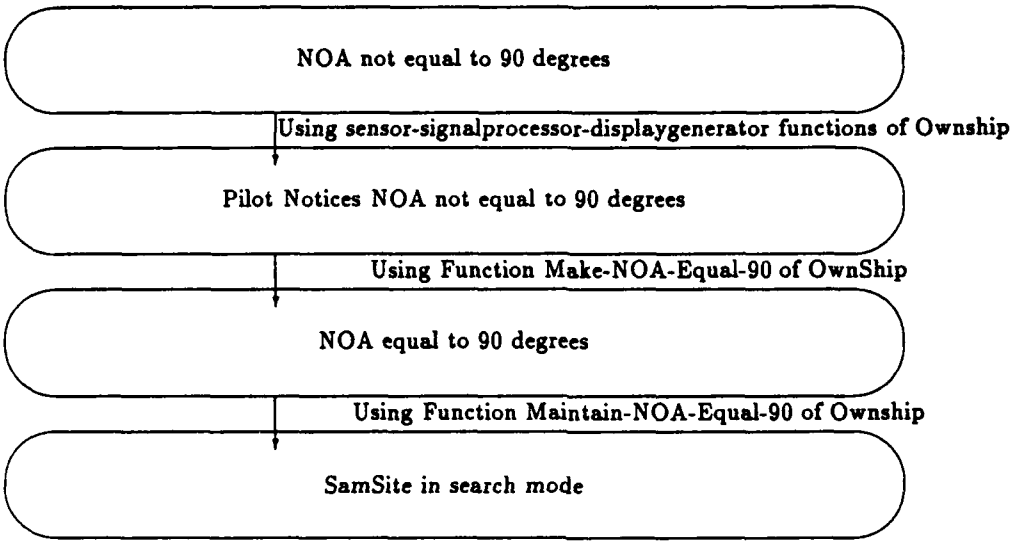


Figure 4: CPD: Constant-Distance-Behavior(fixedground)

that the PopupSamSite is in track mode. This awareness of the pilot then leads to the pilot's using a function, Break-Track, to cause the result state of the PopupSamSite losing track mode.

One virtue of the FR and CPD modeling approach is that we are able to skip over some processes for which we do not have models at a complete level of detail. We do not know the precise retinal and cortical events leading to pilot registration of a situation, for example. Of course, we may know some behavioral parameters for this process (such as average response time). Within the FR and CPD approach, such variables of interest, and equations for computing them when known, are normally attached to the links and state nodes that are marked by the relevant qualitative states and transitions.

In Figure 3, we provide a function frame for the Break-Track function that we have just mentioned. To aid in seeing how a functional decomposition is produced, it is useful at this point to further describe our domain. The specific leaf plan of a possible PA plan-goal graph that we are capturing involves a safety plan, called a doppler notch maneuver, that works in the following way. We suppose there is a particular ground based missile and radar configuration that employs a doppler radar that can be "tricked" by flying in a circle around the site and maintaining constant distance from the radar installation. We now wish to translate this maneuver into the underlying causal processes that carry it out. We next expand our representation preferentially to consider the decomposition of the Constant-Distance-Behavior CPD indexed in the By slot of Figure 3.

Figure 4 shows the expansion of the pilot behavior that goes into the doppler notch maneuver. To achieve a constant distance effect, the pilot thinks of flying in an arc that maintains the radar base at an angle of ninety degrees off

Device: Ownship
Function: Maintain-NOA-Equal-90
If: (NOA within 90 +/- 7.5 degrees)
Maintains: (NOA within 90 +/- 7.5 degrees)
By: (PrecisionFixed)
Decomposition knowledge comments:

Figure 5: Function Maintain-NOA-Equal-90(PopupSamSite) of Ownship

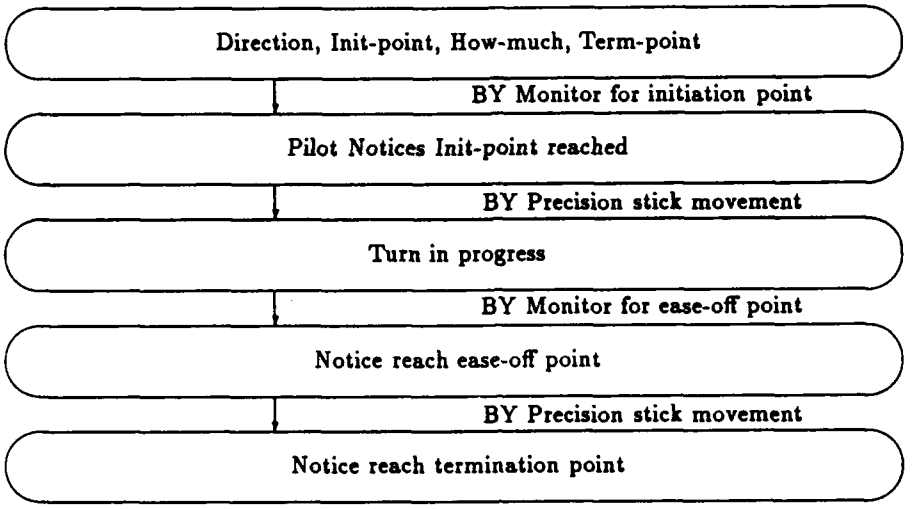


Figure 6: CPD: Precision-Fixed-Maneuver

the line through the plane's nose through its tail. We refer to this orientation mode as NOA, for nose-off-angle. The maneuver is decomposable into a state of being in a NOA unequal to ninety degrees leading to a state of noticing that fact, followed by making and then maintaining the NOA equal to ninety degrees.

Figure 5 represents the function frame for the function Maintain-NOA-Equal-90. The If slot states that this function is only appropriate when ownship is already close to a NOA of 90 degrees (e.g., +/- 7.5 degrees). The By slot states that this function is accomplished by a class of maneuver that we call a PrecisionFixed maneuver. Such a maneuver is typically a very deliberate and patiently executed maneuver with a concern for steady and exacting tolerances.

Figure 6 further decomposes the PrecisionFixed type of maneuver. Such a maneuver starts out in a state of knowing information such as the direction of the maneuver, the points at which to initiate and terminate the maneuver, and how much or how strong of a turn the maneuver will involve. The pilot then must monitor for the intitiation point and start the maneuver upon reaching that point. The pilot initiates and then maintains the maneuver by making precise movements of the stick until he nears the termination point, at which time he eases out of the turn by another precision stick movement.

Figure 7 describes the function of making a precision stick movement. This function requires that the stick must initially be close to the desired position. It also requires that the pilot has perceptual feedback at a sufficiently fine-grained and accurate level to permit the precision maneuver. For example, if NOA were only displayed in 10

Device: Stick
Function: Precision-stick-movement
If: close to desired "direction"
Makes: equal to desired "direction"
Provided: accuracy of instrumentation within range of "close"
By: small gradual movement of stick guided by pilot's perceptual-motor skills
Decomposition knowledge comments:

Figure 7: Function Precision Stick Movement Function

degree increments then it will be difficult for the pilot to make the turn within a tolerance of a few degrees. Similarly, the same problem will arise if the NOA information is only accurate to within 10 degrees.

We have now reached a level where we can begin to connect our mission performance model to hardware components and VHDL models of those hardware components. For example, this function could easily include pointers to hardware components that provide information on NOA. In this case such information would probably come from a radar warning reciever. We could use this connection in two general ways. One use would be to specify the accuracy required in NOA readings by this maneuver. This would be deriving a hardware specification from the high level mission-performance model. A second type of use is an inverse of the first. This would be to constrain the performance model by available accuracy limits of different instruments–such as the radar warning reciever in this case.

3. BOTTOM-UP EXAMPLE OF BRIDGING MISSION PERFORMANCE AND DEVICE IMPLEMENTATION LEVELS

The LSPA program was divided into two parts: the Learning System for Tactical Plans (LSTP) which used machine learning techniques to acquire a new plan by observing a pilot-flown example, and the Learning System for Information Requirements (LSIR) which took the newly-learned plan as input and generated a set of information elements a pilot would require in order to perform the plan. The Pilot-Vehicle Interface module of the PA uses the lists of information elements generated by LSIR to dynamically select a set of display formats for presentation in the cockpit based on the set of active plans. LSIR uses several paramters to describe and reason about each information element. Two examples are *responsiveness* and *task rate-of-change*. The responsiveness parameter indicates how quickly an information element changes value in response to a query or command for a change. Task rate-of-change refers to how often the values of a given information element tend to change.

Even though the parameters of LSIR information elements are at a very detailed grain size relative to tactical plans, they involve information at a quite large grain size relative to most existing VHDL models. One of the authors, however, has been a principal designer and implementor of a model using VHDL for systems performance and architectural evaluation of the next-generation cockpit display generator. This VHDL model, known as the Cockpit Display Generator (CDG), was sponsored by the Wright Laboratories Cockpit Technology Directorate. It is being driven primarily by the development of next-generation rotary wing, transport and fighter aircraft.

CDG is the only existing VHDL model we know that comes close to describing avionics devices at an appropriate level for these LSIR parameters. Because of the availability of CDG, we have focused our analyses on information parameter values that are entirely controlled by the graphics processor—primarily those which are aspects of the displays themselves. One such information element is the range setting of the Tactical Situation Display (e.g., 5, 10, 20, 40, or 60 nautical miles). The responsiveness and task rate-of-change parameters for the Tactical Situation Display range setting are only affected by the performance of the graphics processor and the display surfaces themselves.

Knowledge about latency and throughput values for information requirements that are localised in the graphics processor are readily available via simulation of the CDG models. Latency values can be directly used for the LSIR responsiveness parameter. The task rate-of-change parameter can easily be calculated from the VHDL throughput values. Although these two parameters are a limited portion of the total number of parameters that LSIR requires,

our analyses indicate that we should be able to obtain a substantial proportion of the total number of parameters once more detailed VHDL models become available. We expect that VHDL models of system level devices will become relatively common over the next 10 to 15 years[26].

A major limitation of the present CDG model is that it is a VHDL *performance* model. That is, it only models processing times. This is in contrast to VHDL *functional* models which will represent and reason about the values that are produced in addition to simply modeling how fast inputs are processed. However, even with pure performance models, latency and throughput values can be derived for information requirements not exclusively associated with the graphics processor. This can be accomplished by "stringing" VHDL models together. For example, the latency associated with a commanded change to the mode of the aircraft's FLIR will be the sum of the latencies from the pilot's button press, through the mission computer, to the FLIR itself, then from the FLIR to the mission computer to the graphics processor to the display. The throughput for value changes will be the limiting throughput along the same path.

4. TOP-DOWN EXAMPLE OF BRIDGING MISSION PERFORMANCE AND DEVICE IMPLEMENTATION LEVELS

The integrated representation we presented in section 2 was helpful in organizing our thinking about plans, information elements, and their connection to the VHDL device models. This representation, however, is not strictly necessary for implementing the bottom-up use of the CDG model we described for providing parameters on LSIR information elements. In this section we describe another possible connection between the mission performance systems and device level descriptions in VHDL. We refer to this as a *top-down* connection because we propose to use the high level mission performance model to derive parameters for use in the VHDL model. In particular, we suggest an approach to generating testbenches for VHDL simulations.

Note that the VHDL simulations we described above need to be sensitive to the relevant plans with which the information elements are associated. For example, task rate-of-change of a velocity display (how often the values of the display change) will be very different for maneuvers that involve fast accelerations versus maneuvers that only involve cruising at constant velocities. Such plan dependencies require that the simulation include the relative load levels on the graphics processor imposed by other activities which are known to be ongoing during the plan. For example, during a Doppler Notch maneuver, high-g turns mean that the digital map equipment on the aircraft will be required to go through a large amount of terrain data and be updated frequently. This implies a relatively high load on the graphics processor compared to most other plans.

A simulation of the graphics processor load might be determined in accordance with the following scheme for automatic generation of a testbench of the graphics processor avionics subsystem. First, instantiate a high level plan for a given situation. This could easily be done by using flight simulators such as were used on the PA and LSPA programs. For each plan that is operative in the generated situation, (and perhaps a background load that is typical for the system) a list of display-relevant events must be generated. Rules for inferring such an event list already exist as part of the LSIR system. New rules must be added that will translate these event lists into data flow estimates for the VHDL model. These estimates will provide the initial conditions for the simulation run of the VHDL model.

The Functional Representation framework would be used to to control the computation and temporal ordering of the data flow estimates. The idea is to attach methods to the Functional Representation that compute data flow estimates. These computations will depend on their location in the representation and instantiations of variables for activated plans. A traversal of the representation that respects the causal transition order should produce the proper temporal ordering on the data flow specifications.

5. CONCLUSIONS AND FUTURE WORK

We have reported preliminary work investigating whether an AI high-level mission performance model can usefully connect to and share information with a low-level VHDL model of hardware components. To date, our work has

mainly been an analysis of the problem based on two AI high-level mission performance models, the Pilot's Associate and Learning Systems for Pilot Aiding systems, and a system level VHDL model of a graphics display processor, the CDG model. In this paper we have sketched an example of a connection between such systems using Functional Representation. We have also described an example in which information for the mission performance model is obtained from a VHDL device model and another example in which testbench parameters for the VHDL model can be generated from the mission performance model.

These examples are all in early stages of development. The examples of reasoning between the two types of systems presently only make allusions to using the Functional Representation that we described. The value of this integrated representation has so far primarily been in organising our conceptualisation of the problem domain. We believe, however, that the only notion broad enough to encompass all connections of hardware, software, human behavior and external world environment at the mission performance level is that of functionality. Thus, we anticipate that such a representation will become increasingly crucial to usefully bridging the mission performance and device implementation levels as this work proceeds.

In this paper we have defined our problem and approach and have sketched a representation for a small example. We have also given two examples of potential uses of this representation. Our next steps will include implementing examples of these top-down and bottom-up uses. Our future plans also include scaling-up our representations to handle more scenarios and the different perspectives that will be required by the different modules such as situation assessment and system status.

6. ACKNOWLEDGMENTS

This paper is based on work performed for the Learning Systems Pilot Aiding contract from the Wright Laboratory (Contract Number F33615-88-C-1739). We are pleased to acknowledge inputs from Todd Carpenter, Gurdial Saini, Jerry Covert, Christine Reynolds, and Jerry Meyers. Dale Moberg gratefully acknowledges partial support under DARPA–AFOSR Contract F49620-89-C-0110, Design and Diagnosis Problem Solving with Multifunctional Technical Knowledge Bases.

7. REFERENCES

References

[1] G DeJong Personal communication, March, 1992.

[2] Y. Iwasaki and B. Chandrasekaran. Design verification through function- and behavior-oriented representations: Bridging the gap between function and behavior. To appear in *Proc. Intern. Conf AI in Engineering*, Pittsburgh, PA., 1992.

[3] Korda, S.T., Josephson, J.R., Moberg, D. "Combining functional representation and structure-based models for smarter simulation." LAIR Technical Report, 1992.

[4] Larson, R.A. "An architecture for explicit representation of cause and function in discrete event simulation modeling," M.S. Thesis, University of Minnesota, 1992.

[5] Tom Mitchell, Sridhar Mahadevan, Louis Steinberg "LEAP: A Learning Apprentice for VLSI Design" Proceedings of the 9th International Joint Conference on AI LA,CA August 18-23, 1985 vol. 1 pp 573-580

[6] R. A. Marcotte, M.J. Neiberg, R. L. Piassa and L. J. Holtsblatt. "Model-Based Diagnostic Reasoning Using VHDL" In *Performance and Fault Modeling with VHDL*, J. M. Schoen (ed.), Prentice Hall 1991.

[7] Richard H. Lathrop, Robert J. Hall Robert S. Kirk "Functional Abstraction from Structure in VLSI Simulation Models" Proceedings of the 24th Design Automation Conference 1987, pp 250-256

[8] S.B. Banks & C.S. Lizza "Pilot's Associate: A Cooperative, Knowledge-Based System Application," *IEEE Expert*, Vol. 6, No. 3, June 1991, pp. 18 - 29.

[9] Levi, K.R., & Miller, C. "Automated Acquisition of Plans and Information Requirements in an Intelligent Agent Architecture," Ninth International Machine Learning Conference, Workshop on Architectures for Suporting Machine Learning and Knowledge Acquisition, Aberdeen, Scotland, July, 1992.

[10] K. Levi, D.P. Perschbacher, M. Hoffman, C.A. Miller, V.L. Shalin, B.B. Druhan An Explanation-Based Learning Approach to Knowledge Compilation: Application to Pilot's Associate *IEEE Expert*, Vol. 7, No. 3, June 1992, pp. 44 - 51.

[11] K.R. Levi, V.L. Shalin, E.D. Wisniewski, & P. Scott, "An Analysis of Machine Learning Applications for Pilot-Aiding Expert Systems," Final report for contract no. F33615-86-C-1125. AFWAL Technical Report TR-87-1147, 1987.

[12] K.R. Levi, V.L. Shalin, & D.L. Perschbacher, "Automating Acquisition of Plans for an Intelligent Assistant by Observing User Behavior," *International Journal of Man-Machine Studies*, Vol. 33, 1990, 489 - 503.

[13] Shalin, V.L., Wisniewski, E.D., Levi, K.R., & Scott, P.D. A formal analysis of machine learning for knowledge acquisition *International Journal of Man-Machine Studies, 29*, 1988, 429-446.

[14] G. DeJong & R. Mooney "Explanation-based Learning: An Alternative View," *Machine Learning* Vol. 1, 1986, pp. 145 - 176.

[15] D. Allemang. *Understanding Programs as Devices.* PhD thesis, Ohio State University, 1990.

[16] B. Chandrasekaran. Design problem solving: A task analysis. *AI Magazine*, 11(4):59–71, 1990.

[17] A. Goel and B. Chandrasekaran. Functional representation of designs and redesign problem solving. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1388–1394, August 1989.

[18] A.M. Keuneke. Device representation: the significance of functional knowledge. *IEEE Expert*, April:22–25, 1991.

[19] M. Pegah, W. E. Bond, and J. Sticklen. Representing and Reasoning about the Fuel System of the McDonnell Douglas F/A-18 from a Functional Perspective. *IEEE Expert*, To appear.

[20] V. Sembugamoorthy and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem-solving systems. In J. Kolodner and C. Reisbeck, editors, *Experience, Memory, and Reasoning*, pages 47–73. Lawrence Erlbaum Associates, 1986.

[21] J. Sticklen, W. E. Bond, and D. C. St. Clair. Introducing functional reasoning into the aerospace domain. In *Proceedings of the 4th Conference on AI Applications for Space, NASA.*

[22] J. Sticklen, A. Kamel, and W. E. Bond. A model-based appproach for organizing quantitative computations. In *Second Annual Conference on AI Simulation and Planning in High Autonomy Systems*, Orlando, FL, (1991).

[23] B. Chandrasekaran Models vs Rules, Deep vs Compiled, Content vs Form: Some Distinctions in Knowledge Systems Research *IEEE Expert*, 1991

[24] Dale Moberg and John Josephson. Diagnosing and fixing faults in theories: Appendix: An implementation note. In *Computational Models of Discovery and Theory Formation*, 1990.

[25] Mike Weintraub. An Explanation-Based Approach to Assigning Credit. Dissertation, Advisor: Tom Bylander, The Ohio State University, 1991.

[26] Rose, F. and Carpenter, T. VHDL's Role in Knowledge Acquisition. Proceedings of the Eleventh Applications of Artificial Intelligence Conference, AAI-XI: Knowledge-Based Systems in Aerospace and Industry.